



# CRITICAL PROBLEMS IN VERY LARGE SCALE COMPUTER SYSTEMS

**Semiannual Technical Report for the period October 1, 1987 to March 31, 1988**



Massachusetts Institute of Technology
Cambridge, Massachusetts 02139




| | | |
|---|---|---|
| Principal Investigators: | Paul Penfield, Jr. | (617) 253-2506 |
| Individual Investogators: | Anant Agarwal | (617) 253-1448 |
| | William J. Dally | (617) 253-6043 |
| | Lance A. Glasser | (617) 253-4677 |
| | Thomas F. Knight, Jr. | (617) 253-7807 |
| | F. Thomson Leighton | (617) 253-3662 |
| | Charles E. Leiserson | (617) 253-5833 |
| | Jacob K. White | (617) 253-2543 |
| | John L. Wyatt, Jr. | (617) 253-6718 |


*This research* was sponsored by Defense Advanced Research Projects Agency (DoD), through the Office of Naval Research under Contract No. N00014-87-K-0825.



Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138

# TABLE OF CONTENTS



Selected Publications (starting after page 15)

* W. J. Dally and P. Song, "Design of a Self-Timed VLSI Multicomputer Communication Controller," Proceedings, International Conference on Computer Design, ICCD-87, Port Chester, NY, October 5-8, 1987, pp. 230-234. Also, MIT VLSI Memo No. 87-423, November, 1987.

* W. J. Dally, "A Fine-Grain, Message-Passing Processing Node," Proceedings of the 1987 Workshop on Algorithm, Architecture and Technology Issues in Models of Concurrent Computation, October, 1987. Also, MIT VLSI Memo No. 87-421, November, 1987.

J. L. Wyatt, Jr., and D. L. Standley, "A Method for the Design of Stable Lateral Inhibition Networks that is Robust in the Presence of Circuit Parasitics," to appear in Proceedings, 1987 IEEE National Conference on Neural Networks in Information Processing, Denver, CO, November 8-12, 1987. Also MIT VLSI Memo No. 88-429, January, 1988.

* W. J. Dally, "Concurrent Computer Architecture," Proceedings of Symposium on Parallel Computations and Their Impact on Mechanics, December, 1987. Also, MIT VLSI Memo No. 87-422, November, 1987.

* T. Leighton, C. Leiserson, B. Maggs, S. Plotkin, and J. Wein, "Lecture Notes for 18.435/6.848 Theory of Parallel and VLSI Computation -- Fall 1987," MIT LCS Research Seminar Series #1, March, 1988.

* T. Leighton, C. Leiserson, B. Maggs, S. Plotkin, and J. Wein, "Lecture Notes for 18.436/6.849 Advanced Parallel and VLSI Computation -- Spring 1987," MIT LCS Research Seminar Series #2, March, 1988.

* K. Kundert, J. White, and A. Sangiovanni-Vincentelli, "A Mixed Frequency-Time Approach for Finding the Steady State Solution of Clocked Analog Circuits," 1988 Custom Integrated Circuits Conference, Rochester, NY, May 16-18, 1988. Also, MIT VLSI Memo No. 88-444, March, 1988.

P. R. O'Brien, J. L. Wyatt, Jr., T. L. Savarino, and J. M. Pierce, "Fast On-Chip Delay Estimation for Cell-Based Emitter Coupled Logic," to appear in Proceedings, 1988 IEEE International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also MIT VLSI Memo No. 88-436, February, 1988.

J. L. Wyatt, Jr., and D. L. Standley, "Circuit Design Criteria for Stable Lateral Inhibition Neural Networks," to appear in Proceedings, 1988 IEEE International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-439, March, 1988.

* D. Smart and J. White "Reducing the Parallel Solution Time of Sparse Circuit Matrices using Reordered Gaussian Elimination and Relaxation," to appear in Proceedings, 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-440, March, 1988.

* M. Reichelt, J. White, J. Allen and F. Odeh, "Waveform Relaxation Applied to Transient Device Simulation," to appear in Proceedings, 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-441, March, 1988.

* S. Bhatt, F. Chung, J. Hong, T. Leighton, and A. Rosenberg, "Optimal Simulations by Butterfly Networks," to appear, 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, May 2-4, 1988. Extended abstract issued as MIT VLSI Memo No. 87-427, November, 1987.

W. J. Dally, "Performance Analysis of *k*-ary *n*-cube Interconnection Networks," MIT VLSI Memo No. 87-424, November, 1987.

T. Leighton and E. Schwabe, "Space-Efficient Queue Management Using Fixed-Connection Networks," MIT VLSI Memo No. 87-426, November, 1987.

* J. Katzenelson and R. Zippel, "Software Structuring Principles for VLSI CAD," MIT VLSI Memo No. 88-438, March, 1988.



---

* Abstract only. Complete version available from Microsystems Research Center, Room 39-321, MIT, Cambridge, MA 02139; telephone (617) 253-8138.

# RESEARCH OVERVIEW

This is the first semiannual report on this contract. Most of the work done under the preceding contract, entitled "A Coherent VLSI Design Environment," has been reported in prior years; however some of the papers listed here and a little of the work pertains to that contract, which formally expired December 31, 1987.

The purpose of the present contract is to investigate limiting technologies for a very large computer system, one which, if built during the mid 1990's, would be so large that the nation could only afford one or two of them. Extrapolation of hardware technology trends suggests that the following rough parameters would be achievable:

| | |
|---|---|
| Computational Power | $4\times10^{15}$ FLOPS |
| Power Dissipation | 50+ MW |
| Memory | $3\times10^{15}$ bits |
| Bisection Capacity | $4\times10^{18}$ bits/sec |
| Number of Physical Processors | $3\times10^{8}$ |
| Number of Virtual Processors | $10^{12}$ |
| Size | 14-story building |

The purpose of this contract is to investigate plausibility, which is defined as the step before feasibility. Once plausibility is demonstrated, then separate efforts at demonstrating feasibility and then design and manufacture would be appropriate. It is estimated that the cost of such a "building-size computer" would be in excess of $1,000,000,000.

Six critical areas were identified, and work is in progress in each of these areas. The six areas, and the MIT faculty members who are working in each of the areas, are listed in the chart on the next page. The format of this report is based on this table.

The work in circuits is a combination of improved techniques in waveform bounding, and new ideas in LU factorization and relaxation methods. It has been found that in the context of highly parallel computation, the Gauss-Jacobi technique is never inferior to the Gauss-Seidel technique.

Prototype circuits for a proposed processing element have been fabricated, and are under evaluation.

The investigation of communications topology and related architectural concerns centers around two projects. In one of them, the Message-Driven Processor is the object of study. A prototype network design frame has been fabricated and tested. The study of cache techniques and dictionary schemes has just begun, with the addition of Prof. Anant Agarwal to the faculty.

In the systems software area, the effort centers on an operating system for the Message-Driven Processor. Three major paradigms for resource management have been found: object replication, proactive mechanisms, and elimination of dynamic bottlenecks.

The work on algorithms involves highly parallel algorithms for a variety of purposes, including graph theory. Many new results are given in this section and in the various publications.

The application area so far has centered around parallel simulation of a large analog VLSI chip. The chip considered is one that can be part of a smart sensor. No current simulation tools can handle circuits of such complexity and scale.

The principal activities and contributing faculty are listed here:

| | |
|---|---|
| Circuits | Lance A. Glasser<br>Thomas F. Knight, Jr.<br>Paul Penfield, Jr.<br>Jacob K. White<br>John L. Wyatt, Jr. |
| Processing Elements | William J. Dally |
| Communications Topology and Routing Algorithms | Anant Agarwal<br>William J. Dally<br>F. Thomson Leighton<br>Charles E. Leiserson |
| Systems Software | William J. Dally |
| Algorithmic Models | F. Thomson Leighton<br>Charles E. Leiserson |
| Applications | Charles E. Leiserson<br>Jacob K. White<br>John L. Wyatt, Jr. |

## CIRCUITS

Prof. John Wyatt, D. L. Standley, and P. O'Brien have been finishing up our previous project on CAD and beginning a new one on parallel simulation methods for regular arrays. The goal of this last phase of our CAD project has been to extend the waveform bounding results that Penfield et. al. developed for MOS circuits so that they work for high-speed ECL as well. Our recent work consists of two major parts:

- macromodelling of ECL logic gates acting both as drivers and as loads, and
- delay estimation for individual nets using the gate macromodel parameters and RC tree models for metal interconnect.

The success of the macromodelling approach relies on repetitive use of a library of modelled cells. A fixed computational cost (several mainframe CPU hours per cell) is paid to obtain parameter values for the simplified macromodels. The resultant timing estimates are typically within 5% to 10% of SPICE and are obtained with roughly 1000 times less CPU time per run. This work has taken a very practical turn and has been extensively tested on an industrial ECL process and cell library. It is now in use in American industry.

Prof. Jacob White is investigating the possibility of accelerating the transient simulation of MOS devices by using waveform relaxation. The WR algorithm is being applied to the sparsely-connected system of algebraic and ordinary differential equations in time generated by standard spatial discretization techniques. It was proved that the WR contracts in a uniform norm on a model of the device simulation problem, and the result was verified on a one-dimensional experiment*. The implementation of the method for 2-D device simulation is in progress.

LU factorization of Circuit simulation matrices is difficult to parallelize, in part because methods like parallel nested dissection are ineffective due to the difficultly of finding good separators. We are investigating a more direct approach to exploiting the parallelism in sparse matrix solution. In particular, we are examining the interaction between sparse matrix data structures and computer memory structure, to see how to store sparse matrices for effective parallel computation. One interesting result is that it is possible to store matrices in a scattered form which allows for access of the sparse entries by fast indexing, in only three times the storage. This data structure is static, constructed once the matrix structure is known, and is therefore very attractive for parallel implementation.

The results indicating the optimality of Gauss-Jacobi over Gauss-Seidel relaxation on infinitely many parallel processors has been extended, mostly to connect the spectral radius of the iteration matrices to their graphical properties†. In addition, the results are being extended to the waveform relaxation case, where this limiting result seems to show on as few as eight processors.

We are trying to improve the reliability of relaxation methods by extracting lines, or banded matrices, from a given sparse matrix, solving the bands directly, and relaxing on the rest of the matrix. This approach is efficient because band matrices can be solved in $\log n$ time on $n$ processors, and is more reliable than standard relaxation, because "less" relaxation is being used. The idea has been tested on circuit simulation matrices and converges faster and more often than standard relaxation on all examples tried so far. We are

---

*M. Reichelt, J. White, J. Allen and F. Odeh, "Waveform Relaxation Applied to Transient Device Simulation," 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988.

† D. Smart and J. White "Reducing the Parallel Solution Time of Sparse Circuit Matrices using Reordered Gaussian Elimination and Relaxation," 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988.

presently working on how to select the ordering of the matrix to best exploit the direct solution of the band, and to automatically select the band size. In addition, we are working on implementing the method on the Connection Machine.

A new technique for simulating switched-capacitor filters and switching power supplies in steady state was developed *. The idea is based on simulating selected cycles of the high-frequency clock accurately with a standard discretization method, and pasting together the selected cycles by computing the low frequency behavior with a truncated Fourier series. If carefully constructed, the nonlinear system that must be solved for the Fourier coefficients is almost linear and can be solved rapidly with Newton's method.

The WRN method with mesh refinement is more parallelizable than standard WR methods, because most of the computation for all the timesteps in a given waveform can be computed in parallel. Recent work on this method has been to show that the waveform Newton algorithm converges globally when applied to circuits with nonlinear capacitors †.

The model used in conventional device simulation programs is based on the drift-diffusion model of electron transport, and this model does not accurately predict the field distribution near the drain in small geometry devices. This is of particular importance for predicting oxide breakdown due to penetration by "hot" electrons. Two approaches for improving the accuracy of the drift-diffusion model of electron transport are under investigation. The first approach is to take an additional moment of the Boltzmann equation, which yields a system of equations for electron transport that is similar to the drift-diffusion model, but includes the electron energies. The model is referred to as the hydrodynamic model and has been implemented in several simulators.

However, the discretization techniques used in these simulators create instabilities in certain cases. In addition, the hydrodynamic model relies on knowing electron energy relaxation times, which are not easily measured. For this reason, the Monte Carlo method is also being investigated. This is a microscopic method in which many particles (thousands) are tracked as they go through random scatterings. This method is computationally very expensive, but can model very detailed physics. It is, however, an ideal candidate for parallel computation as the particles can be tracked independently, being coupled only through the electric field. Implementation of a Monte Carlo simulator on the Connection Machine is currently under investigation.

---

*K. Kundert, J. White, A. Sangiovanni-Vincentelli, "A Mixed Frequency-Time Approach for Finding the Steady State Solution of Clocked Analog Circuits," 1988 Custom Integrated Circuits Conference, Rochester, NY, May 16-18, 1988.

† R. Saleh, J. White, A. Sangiovanni-Vincentelli, and A. R. Newton, "Accelerating Relaxation Algorithms for Circuit Simulation Using Waveform-Newton and Step-Size Refinement," submitted to IEEE Transactions on CAD.

## PROCESSING ELEMENTS

Prof. Dally and Stuart Fiske have been developing the Reconfigurable Arithmetic Processor (RAP), which, as described in our last report, is an experiment in reducing the I/O bandwidth required by VLSI arithmetic chips. The RAP exploits the locality inherent in equations by chaining together a number of nibble-serial arithmetic units using a switch that is reconfigured every major cycle. The use of nibble-serial arithmetic allows us to build a switch in 1/64th the area required by a full parallel switch, and to avoid the pipeline latency of parallel pipelined arithmetic units.

Stuart Fiske has completed a simulation study of six benchmark numerical applications. The average performance of the RAP on these benchmarks is 9MFLOPs, 45 % of the 20MFLOPs peak performance.

Our fixed point RAP prototype has been tested. The initial version was missing a ground connection to the output register decoder. This bug prevented us from reading out any output register except R0. Working around this bug (using R0) we verified proper operation of the input registers, the crossbar switch, the multipliers, and the adders. A revised chip, correcting the problem, has been fabricated and tested.

Stuart Fiske is currently designing a 64-bit floating-point RAP test chip. He has completed the circuit design of a 64-bit, nibble-serial, floating-point adder/subtracter, a major component of the floating-point RAP.

## COMMUNICATIONS TOPOLOGY AND ROUTING ALGORITHMS

The Message Driven Processor (MDP) project involves Prof. Dally, Andrew Chien, Soha Hassoun, Waldemar Horwat, Bill Song, Brian Totty, and Scott Wills. A number of changes were made to the MDP architecture over the past six months. Following a visit from Intel Engineer Paul Carrick, a bypass path was added to the data path to improve performance. Also, the controller was redesigned and the trap handling logic was changed to more efficiently handle synchronization using futures. A behavioral register transfer level (RTL-B) simulation has been implemented to test these changes. We have also implemented a structural RTL simulator (RTL-S) to debug the MDP logic equations before drawing detailed schematics of the logic.

Our prototype network design frame chips have been tested. The initial 1D test chip functioned correctly except for a subtle timing bug that occasionally causes a decrementer malfunction. Tests of the prototype router have verified the remainder of the router logic, the low-voltage I/O pads, and the use of a single wire for request and acknowledge. A revised router chip has been submitted for fabrication.

The current research by Prof. Agarwal involves highly efficient large-scale shared-memory multiprocessing. Memory traffic over the network is one of the primary concerns in such systems. To minimize the traffic to memory over the communication network, we intend using large caches. Unlike previous large-scale efforts (e.g. the IBM RP3), we will cache shared memory blocks in addition to private blocks. Caching not only reduces the latency of memory accesses but also substantially increases the effective memory bandwidth. To maintain cache coherency, we intend using a modified directory scheme. This scheme does not need a shared bus as the communication medium; shared buses are necessary in snoopy-cache systems (e.g., the Berkeley SPUR) and tend to limit the scalability of the multiprocessor.

He have already shown that our directory schemes are as efficient as snoopy cache methods for maintaining cache consistency. In addition, directory schemes have the potential to scale far beyond the limits of snoopy caches. Some early indication of their scalability was provided by simulations using address traces of small multiprocessors (4 processors). Our immediate goal is to evaluate the scalability of these directory schemes for a much larger number of processors.

## SYSTEMS SOFTWARE

The JOSS operating system for the Message-Driven Processor (MDP), described in our last report, has been extended by adding new code for heap compaction, fast context allocation, late binding method activation, object migration and method caching.

Prof. Dally, Andrew Chien, and Waldemar Horwat have implemented a new Concurrent Smalltalk (CST) programming system. This system will be used to experiment with fine-grain message-passing application programs. The initial implementation, written in Common Lisp, consists of a prefix-CST compiler that generates message-passing intermediate code (I-Code), an I-code interpreter, and an instrumentation package. The system runs on Symbolics 36XX Lisp Machines and Sun workstations. Using this system, we have written and simulated many simple programs to gather statistics about fine-grain concurrent programs. We are currently working on coding a few realistic benchmark programs (e.g., a particle-in-cell (PIC) simulation). To determine the suitability of the MDP to execute this class of programs, Waldemar Horwat has written a code generator that converts I-Codes to MDP assembly language.

We have identified several key paradigms for resource management (RM) in a message-passing computer system that represent generalizations of RM schemes that have appeared in other computer systems. We have uncovered three major paradigms: object replication, proactive vs. reactive mechanisms, and elimination of dynamic bottlenecks.

Object replication techniques have been used for many years in caching immutable objects (instructions or read-only data). In this case, replication is done on the basis of cache lines. Cache coherency protocols in multiprocessors support replication of mutable objects. In these systems, multiple readers but only one writer is allowed for a given memory location. Again, the replication is done on the basis of a cache line. We have generalized the problem and are considering two cases: 1) replication of immutable objects and 2) replication and consistency control of mutable objects. In an object oriented model, these decisions are made on a per object basis (this makes more sense) and we can make these decisions dynamically.

Resource management techniques can be thought of as proactive or reactive. Proactive techniques attempt to anticipate future events and provide for them. Reactive techniques respond to actual needs. Proactive techniques are less well understood, but promise significantly better performance.

The elimination of dynamic bottlenecks is another important paradigm in concurrent machines. Such bottlenecks appear to occur often in code loading. Reduction operations (accumulations or histograms) can also give rise to dynamic bottlenecks. Several software schemes based on combining have been designed to reduce the impact of such bottlenecks.

In all of these paradigms, optimization depends on information about a program's run time behavior. Some of this information may be gleaned by sophisticated compilation techniques. However, we feel that there is often a need for the programmer to make some assertions about the program.

## ALGORITHMS

Mark Newman, Johan Hastad and Prof. Leighton continued their work on techniques for using a hypercube even when a significant portion of its nodes and edges are faulty. Previously, a fast, local and deterministic algorithm was shown to reconfigure the working nodes of a hypercube into a fully functioning hypercube of only one lower dimension even when up to half of the nodes of the original hypercube contain random faults. These reconfigurations enabled the hypercube's working nodes to simulate a wide variety of constant degree networks with only constant slowdown, again with the failure of up to half of the nodes. For example, even if half of the nodes of the hypercube have random malfunctions, the hypercube can still simulate every binary tree of the same size with constant slowdown.

Recently, these results were extended to include wire faults, as well as scenarios where far less than half of the hardware malfunctions. Progress was also made on the problem of reducing the congestion encountered during the simulation of a functioning hypercube by a faulty hypercube. It is hoped that the new work will lead to a simulation algorithm that has constant congestion independent of the size of the hypercube.

Eric Schwabe and Prof. Leighton are continuing their work on space-efficient techniques for queue management in very large scale networks. One of the main difficulties in designing algorithms for current large scale parallel machines is making sure that the capacities of the local memories are not exceeded. Schwabe and Leighton have made substantial progress towards solving this problem by designing a general scheme for dynamically reorganizing memory so that local memory constraints are never exceeded provided that global memory constraints are not exceeded. The latter constraint is, of course, much easier to insure for large scale parallel machines where a very large amount of memory is distributed in very small chunks among a large number of processors. The new scheme is simple, real-time, space-efficient, deterministic, and transparent to the programmer. It requires only that the total hardware of the system (i.e., wires and total memory) exceed the number (not size) of the local memories being managed by a logarithmic factor. In return, the scheme guarantees an arbitrarily high percentage utilization of the total memory. The scheme is specifically designed for use with hypercube-related networks, and works well in both worst-case and average-case settings. Results of this work are reported in *.

Working with coauthors from Yale, Bellcore, and UMass-Amherst, Prof. Leighton is continuing his research in network simulations. The speed with which one network can simulate another is one good indication of the relative computing power of the networks. For example, the hypercube can simulate a mesh of the same size without delay, but not vice-versa. Recently, Prof. Leighton and his coauthors showed that the hypercube can simulate just about every other network proposed for parallel computation with only constant slowdown. Included are: all binary trees, the X-tree, the pyramid networks, all grid networks, meshes of trees, cube-connected-cycles, and the butterfly. There remained the question of whether or not other networks such as the butterfly had the same property. Recently, Prof. Leighton and coauthors showed that this was not the case by developing general lower bound techniques for showing that one network cannot simulate another very efficiently. For example, they proved that networks like the butterfly and shuffle-exchange graph cannot simulate networks like the X-tree, pyramid graph or mesh without a $\log N$ factor slowdown, the worst possible. Results of this work can be found in †.

---

*T. Leighton and P. Shor, "Tight Bounds for Minimax Grid Matching, with Applications to the Average Case Analysis of Algorithms," Combinatorica, to appear, 1987.

† S. Bhatt, F. Chung, J. Hong, T. Leighton and A. Rosenberg, "Optimal Simulations by Butterfly Networks," 1988 ACM Symposium on Theory of Computing, to appear.

Working under the direction of Prof. Leighton, Seth Mallitz recently resolved an open question concerning the pagenumber of graphs with E edges by showing that every E-edge graph can be embedded in a book with $O(E^{1/2})$ pages and that this is close to optimal. Recently, this work has been extended further to show that the pagenumber of any graph with genus G is at most $O(G^{1/2})$, again the best possible, in general. Determining the pagenumber of a graph has relevance to problems in wire routing and fault tolerance, and has been discussed in previous research reviews. The current work * substantially improves the best known previous bounds from above and below.

Working with Profs. Finkelstein (Northeastern) and Kleitman, Prof. Leighton discovered a method for efficiently realizing permutations of data stored in VLSI chips using bus interconnections with a small number of pins per chip. The work resolves the question left open in the recent work on the same subject by Kilian, Kipnis and Leiserson, and provides tight bounds on the number of pins required to realize permutations with uniform permutation architectures. Results of this research are reported in †.

Prof. Leiserson and Cindy Phillips have extended their previous results for parallel graph contraction. The new results allow any bounded-degree n-vertex graph to be contracted in time $O(\lg n + \lg^2 g)$ on an EREW PRAM where g is the largest genus of any connected component of the graph. The algorithm does not require knowledge of the genus. Cindy Phillips, Guy Blelloch, and Ajit Agrawal and Robert Krawitz of Thinking Machine Corporation have implemented on the Connection Machine a set of techniques for manipulating large dense matrices on hypercube multiprocessors. In particular, they consider the case where there are fewer processors than matrix elements. A set of powerful primitives hide the problem embedding from the user while allowing efficient manipulation of the embedding. Problems such as dense linear systems solving have been sped up by an order of magnitude.

Prof. Leiserson and Bruce Maggs have discovered a class of point-to-point networks that are area universal in the sense that, with high probability, a network in the class with N processors has area $O(N)$ and can simulate in $O(\log N)$ steps each message-step of any shared-bus network of area $O(N)$. The simulation is optimal because a point-to-point network may require $\Omega(\log N)$ steps to simulate one step of a shared-bus network. The area universal networks are based on the fat-trees of Greenberg and Leiserson and the simulation uses a randomized message routing algorithm based on the butterfly routing algorithm of Ranade.

They have also been studying a model for parallel computation called the concurrent-read concurrent-write distributed random-access machine (CRCW DRAM). In a DRAM, the communication requirements of a parallel algorithm can be evaluated. A conservative DRAM algorithm is one whose communication requirements at each step can be bounded by the congestion of pointers of the input data structure across cuts of the network. Previously, they gave a lemma that showed how to "shortcut" pointers in a data structure in an exclusive-read exclusive-write (EREW) DRAM so that remote processors could communicate without causing undue congestion. Recently they discovered a more powerful shortcut lemma for the CRCW DRAM model. Using this lemma one can construct conservative CRCW DRAM algorithms for problems such as finding the minimum cost spanning tree of graph that, with high probability, require $O(\log N)$ steps. The fastest known conservative EREW DRAM algorithms for these problems require $O(\log^2 N)$ steps.

Work is also in progress on several other problems, although definitive results have not yet been achieved. For example, Prof. Leighton is working with Richard Koch on the asymptotic analysis of butterfly routing algorithms like those used by the BBN Butterfly; Prof. Leighton is working with Bruce Maggs and

---

*S. Mallitz, "E-edge Graphs Have Pagenumber $O(E^{1/2})$," to be submitted, March 1988.

† L. Finkelstein, D. Kleitman and T. Leighton, "Applying the Classification Theorem for Finite Simple Groups to Minimize Pin Count in Uniform Permutation Architectures," 1988 Aegean Workshop on Computing, to appear.

Satish Rao on a unified approach to packet routing in arbitrary networks; and Prof. Leighton is working with Satish Rao on the development of a general theory for approximate max-flow min-cut results for multicommodity flows, which will have applications to a variety of problems including VLSI layout, graph bisection, crossing number and packet routing.

# APPLICATIONS

Prof. Wyatt is beginning a new project on the parallel simulation of large, regular analog arrays. The goal is to produce a simulation tool that can be used for the design of smart sensors for machine vision, such as those currently being developed at Carver Mead's laboratory at Caltech. These chips typically consist of large arrays, e.g., from 32×32 to 128×128, of moderately simple analog cells that perform a collective analog computation by communication with nearest neighbors. No currently available simulation tool is of any use on circuits of this size.

Systems of this type have two features that should influence the design of a tailor-made simulator. One is the natural hierarchy: devices in circuits in cells in arrays. The other is the designer's concern with circuit sensitivity in such systems: how do individual component variations affect overall system performance? We are studying ways to adapt the classical adjoint network approach to a hierarchical framework to solve this latter problem with acceptable computational efficiency.

Prof. Leiserson and Jeff Siskind have been investigating efficiency models for Prolog. This past summer, an implementation of Prolog that performed dependency-directed backtracking was described. The methodology used for that implementation was to incrementally unravel a Prolog program into an AND/OR goal tree and represent both the search tree, as well as the unification bindings at its leaf nodes, as a SAT problem in a TMS. This approach required precomputing ALL potential nogoods corresponding to potential unification violations at every unraveling step, clearly a fundamental inefficiency flaw.

An alternative methodology is being pursued in a new implementation. The implementation starts off with a Prolog interpreter whose structure is similar to conventional implementations and whose performance differs from such implementations by only a (presumably small) constant factor. This interpreter is incrementally modified to add mechanisms for doing Selective Backtracking, Lateral Pruning (Caching Nogoods), Non-chronological Retraction, and Constraint Propagation. As each mechanism is added, care is taken to minimize, or at least understand, its effect on efficiency of the baseline system. Each incremental change can be independently enabled or disabled to allow experimental analysis of the benefits accrued by each of the above techniques, in isolation and in combination with the others.

## PUBLICATIONS LIST


W. J. Dally, L. Chao, A. Chien, S. Hassoun, W. Horwat, J. Kaplan, P. Song, B. Totty, and S. Wills, "Architecture of a Message-Driven Processor," Proceedings, 14th Annual Symposium on Computer Architecture, Pittsburgh, PA, June 2-5, 1987, pp. 189-196. Also, MIT VLSI Memo No. 87-420, November, 1987.

P. Agrawal, W. J. Dally, A. K. Ezzat, W. C. Fischer, H. V. Jagadish, and A. S. Krishnakumar, "Architecture and Design of the MARS Hardware Accelerator," Proceedings, 24th IEEE/ACM Design Automation Conference, Miami Beach, FL, June 28-July 1, 1987. Also, MIT VLSI Memo No. 87-419, November, 1987.

P. Agrawal, W. J. Dally and R. Tutundjian, "Logic Simulation Algorithms for Pipelined Hardware Architectures," Proceedings, International Workshop on Hardware Accelerators, September, 1987.

W. J. Dally and P. Song, "Design of a Self-Timed VLSI Multicomputer Communication Controller," Proceedings, International Conference on Computer Design, ICCD-87, Port Chester, NY, October 5-8, 1987, pp. 230-234. Also, MIT VLSI Memo No. 87-423, November, 1987.

W. J. Dally, "A High-Performance VLSI Quaternary Serial Multiplier," Proceedings, International Conference on Computer Design, ICCD-87, Port Chester, NY, October 5-8, 1987, pp. 649-653.

S. Rao, "Finding Near-Optimal Separators in Planar Graphs," Proceedings, 1987 IEEE Symposium on Foundations of Computer Science, Los Angeles, CA, October 12-14, 1987, pp. 225-237.

W. J. Dally, "A Fine-Grain, Message-Passing Processing Node," Proceedings of the 1987 Workshop on Algorithm, Architecture and Technology Issues in Models of Concurrent Computation, October, 1987. Also, MIT VLSI Memo No. 87-421, November, 1987.

P. Agrawal, W. J. Dally, W. C. Fischer, H. V. Jagadish, A. S. Krishnakumar, and R. Tutundjian, "MARS: A Multiprocessor-Based Programmable Accelerator," IEEE Design & Test of Computers, vol. 4, no. 5, pp. 28-36, October, 1987.

J. L. Wyatt, Jr., and D. L. Standley, "A Method for the Design of Stable Lateral Inhibition Networks that is Robust in the Presence of Circuit Parasitics," to appear in Proceedings, 1987 IEEE National Conference on Neural Networks in Information Processing, Denver, CO, November 8-12, 1987. Also MIT VLSI Memo No. 88-429, January, 1988.

W. J. Dally, "Concurrent Computer Architecture," Proceedings of Symposium on Parallel Computations and Their Impact on Mechanics, December, 1987. Also, MIT VLSI Memo No. 87-422, November, 1987.

R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani, "Global Wire Routing in Two-Dimensional Arrays," Algorithmica, vol. 2, pp. 113-129, 1987.

T. Bui, S. Chaudhuri, T. Leighton, and M. Sipser, "Graph Bisection Algorithms With Good Average Case Behavior," Combinatorica, vol. 7, no. 2, pp. 171-191, 1987.

T. Leighton and P. Shor, "Tight Bounds for Minimax Grid Matching, with Applications to the Average Case Analysis of Algorithms," to appear, Combinatorica, 1987.

W. J. Dally, "Fine-Grain Message-Passing Concurrent Computers," Proceedings of Third Conference on Hypercube Concurrent Computers and Applications, Pasadena, CA, January 19-20, 1988.

W. J. Dally and A. A. Chien, "Object-Oriented Concurrent Programming in CST," Proceedings of Third Conference on Hypercube Concurrent Computers and Applications, Pasadena, CA, January 19-20, 1988.

T. Leighton, C. Leiserson, B. Maggs, S. Plotkin, and J. Wein, "Lecture Notes for 18.435/6.848 Theory of Parallel and VLSI Computation -- Fall 1987," MIT LCS Research Seminar Series #1, March, 1988.

T. Leighton, C. Leiserson, B. Maggs, S. Plotkin, and J. Wein, "Lecture Notes for 18.436/6.849 Advanced Parallel and VLSI Computation -- Spring 1987," MIT LCS Research Seminar Series #2, March, 1988.

K. Kundert, J. White, and A. Sangiovanni-Vincentelli, "A Mixed Frequency-Time Approach for Finding the Steady State Solution of Clocked Analog Circuits," 1988 Custom Integrated Circuits Conference, Rochester, NY, May 16-18, 1988. Also, MIT VLSI Memo No. 88-444, March, 1988.

S. Fiske and W. J. Dally, "The Reconfigurable Arithmetic Processor," Proceedings, 15th International Symposium on Computer Architecture, Honolulu, HI, May 30-June 2, 1988.

P. R. O'Brien, J. L. Wyatt, Jr., T. L. Savarino, and J. M. Pierce, "Fast On-Chip De. Estimation for Cell-Based Emitter Coupled Logic," to appear in Proceedings, 1988 IEEE International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also MIT VLSI Memo No. 88-436, February, 1988.

J. L. Wyatt, Jr., and D. L. Standley, "Circuit Design Criteria for Stable Lateral Inhibition Neural Networks," to appear in Proceedings, 1988 IEEE International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-439, March, 1988.

D. Smart and J. White "Reducing the Parallel Solution Time of Sparse Circuit Matrices using Reordered Gaussian Elimination and Relaxation," to appear in Proceedings, 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-440, March, 1988.

M. Reichelt, J. White, J. Allen and F. Odeh, "Waveform Relaxation Applied to Transient Device Simulation," to appear in Proceedings, 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-441, March, 1988.

L. Finkelstein, D. Kleitman, and T. Leighton, "Applying the Classification Theorem for Finite Simple Groups to Minimize Pin Count in Uniform Permutation Architectures," to appear in 1988 Aegean Workshop on Computing.

S. Bhatt, F. Chung, J. Hong, T. Leighton, and A. Rosenberg, "Optimal Simulations by Butterfly Networks," to appear, 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, May 2-4, 1988. Extended abstract issued as MIT VLSI Memo No. 87-427, November, 1987.

S. Mallitz, "E-edge Graphs Have Pagenumber $O(E^{1/2})$," to be published.

R. Saleh, J. White, A. Sangiovanni-Vincentelli, and A. R. Newton, "Accelerating Relaxation Algorithms for Circuit Simulation Using Waveform-Newton and Step-Size Refinement," to be published.

C. E. Leiserson and C. A. Phillips, "Parallel Contraction of Bounded-Degree Graphs," to be published.

A. Agrawal, G. Blelloch, R. L. Krawitz, and C. A. Phillips, "Implementing Data Parallel Operations for Large Matrices on Smaller Parallel Computers," to be published.

## INTERNAL MEMORANDA

L. A. Glasser, "Frequency Limitations in Circuits Composed of Linear Devices," MIT VLSI Memo No. 87-415, October, 1987.

W. J. Dally, "Performance Analysis of $k$-ary $n$-cube Interconnection Networks," MIT VLSI Memo No. 87-424, November, 1987.

T. Leighton and E. Schwabe, "Space-Efficient Queue Management Using Fixed-Connection Networks," MIT VLSI Memo No. 87-426, November, 1987.

J. Katzenelson and R. Zippel, "Software Structuring Principles for VLSI CAD," MIT VLSI Memo No. 88-438, March, 1988.

## TALKS WITHOUT PROCEEDINGS

T. Leighton, "Some Computational Properties of the Hypercube," IBM Watson Research Center, Yorktown Heights, NY, October 1987.

T. Leighton, "Some Computational Properties of the Hypercube," University of Massachusetts, Amherst, MA, November 1987.

T. Leighton, "Some Computational Properties of the Hypercube," University of Maryland, November 1987.

W. J. Dally, "Wire-Efficient VLSI Multiprocessor Communication Networks," MIT-Siemens ZTI Symposium, Munich, Germany, November 1987.

J. L. Wyatt, Jr., "Analog VLSI and Neural Networks - the View from Pasadena," Stochastic Systems Seminar, Department of Electrical Engineering and Computer Science, MIT, November 1987.

T. Leighton and E. Schwabe, "Space-Efficient Queue Management Using Fixed-Connection Networks," MIT VLSI Research Review, Cambridge, MA, December 14, 1987.

G. Blelloch, "Scans as Primitive Parallel Operations," MIT VLSI Research Review, Cambridge, MA, December 14, 1987.

W. J. Dally and P. Y. Song, "Design of a Self-Timed VLSI Multicomputer Communication Controller," MIT VLSI Research Review, Cambridge, MA, December 14, 1987.

J. L. Wyatt, Jr., "Smart Sensors for Vision," Microsystems Industrial Group, MIT, December 1987.

T. Leighton, "Some Computational Properties of the Hypercube," University of Texas, Dallas, TX, January 1988.

T. Leighton, "On the Average Case Analysis of Assignment Algorithms for Problems such as Wafer Scale Integration of Systolic Arrays," AAAS Annual Meeting, February 1988.

W. J. Dally, "Fine-Grain Message-Passing Concurrent Computers," Supercomputing Research Center, Lanham, MD, February 1988.

W. J. Dally, "JOSS: The Jellybean Operating System," AT&T Bell Laboratories, Holmdel, NJ, March 1988.

W. J. Dally, "Wire-Efficient Communication Networks for Multicomputers," Massively Parallel Models of Computation Workshop, Banff, Alberta, Canada, March 1988.

W. J. Dally, "Analysis and Design of Multicomputer Communication Networks," Massively Parallel Models of Computation Workshop, Banff, Alberta, Canada, March 1988.

W. J. Dally, "Architecture of a Message-Driven Processor," Massively Parallel Models of Computation Workshop, Banff, Alberta, Canada, March 1988.

W. J. Dally, "Concurrent Data Structures - Abstractions for Concurrency," Massively Parallel Models of Computation Workshop, Banff, Alberta, Canada, March 1988.



VLSI Memo No. 87-423
November 1987

# DESIGN OF A SELF-TIMED VLSI MULTICOMPUTER COMMUNICATION CONTROLLER

William J. Dally and Paul Song

## Abstract

We describe the design of the network design frame (NDF), a self-timed routing chip for a message-passing concurrent computer. The NDF uses a partitioned data path, low-voltage output drivers, and a distributed token-passing arbiter to provide a bandwidth of 450Mbits/sec into the network. Wormhole routing and bidirectional virtual channels are used to provide low latency communications, less than 2$\mu$s latency to deliver a 216bit message across the diameter of a 1K node machine. To support concurrent software systems, the NDF provides two logical networks, one for user messages and one for system messages, that share the same set of physical wires. To facilitate the development of network nodes, the NDF is a design frame. The NDF circuitry is integrated into the pad frame of a chip leaving the center of the chip uncommitted.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138



VLSI Memo No. 87-421
November 1987

# A FINE-GRAIN, MESSAGE-PASSING PROCESSING NODE

William J. Dally

Abstract

This paper describes a processing node for a fine-grain message-passing concurrent computer. The node consists of a processor, a communication unit, and a memory. To reduce the overhead of message passing and task switching to 5$\mu$s, the node incorporates a send instruction, a fast communication system, hardware message buffering and dispatch, and a general translation mechanism. These mechanisms work together to implement a fine-grain programming system.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

VLSI Memo No. 88-429
January 1988

# A METHOD FOR THE DESIGN OF STABLE LATERAL INHIBITION NETWORKS THAT IS ROBUST IN THE PRESENCE OF CIRCUIT PARASITICS

J. L. Wyatt, Jr. and D. L. Standley

Abstract

In the analog VLSI implementation of neural systems, it is sometimes convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. A serious problem of unwanted spontaneous oscillation often arises with these circuits and renders them unusable in practice. This paper reports a design approach that guarantees such a system will be stable, even though the values of designed elements and parasitics elements in the resistive grid may be unknown. The method is based on a rigorous, somewhat novel mathematical analysis using Tellegen's theorem and the idea of Popov multipliers from control theory. It is thoroughly practical because the criteria are **local** in the sense that no overall analysis of the interconnected system is required, **empirical** in the sense that they involve only measurable frequency response data on the individual cells, and **robust** in the sense that unmodelled parasitic resistances and capacitances in the interconnection network cannot affect the analysis.

# A METHOD FOR THE DESIGN OF STABLE LATERAL INHIBITION NETWORKS THAT IS ROBUST IN THE PRESENCE OF CIRCUIT PARASITICS

J.L. WYATT, Jr and D.L. STANDLEY
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

## ABSTRACT

In the analog VLSI implementation of neural systems, it is sometimes convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. A serious problem of unwanted spontaneous oscillation often arises with these circuits and renders them unusable in practice. This paper reports a design approach that guarantees such a system will be stable, even though the values of designed elements and parasitic elements in the resistive grid may be unknown. The method is based on a rigorous, somewhat novel mathematical analysis using Tellegen's theorem and the idea of Popov multipliers from control theory. It is thoroughly practical because the criteria are local in the sense that no overall analysis of the interconnected system is required, empirical in the sense that they involve only measurable frequency response data on the individual cells, and robust in the sense that unmodelled parasitic resistances and capacitances in the interconnection network cannot affect the analysis.

## I. INTRODUCTION

The term "lateral inhibition" first arose in neurophysiology to describe a common form of neural circuitry in which the output of each neuron in some population is used to inhibit the response of each of its neighbors. Perhaps the best understood example is the horizontal cell layer in the vertebrate retina, in which lateral inhibition simultaneously enhances intensity edges and acts as an automatic gain control to extend the dynamic range of the retina as a whole[1]. The principle has been used in the design of artificial neural system algorithms by Kohonen[2] and others and in the electronic design of neural chips by Carver Mead et. al.[3,4].

In the VLSI implementation of neural systems, it is convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. Linear resistors fabricated in, e.g., polysilicon, yield a very compact realization, and nonlinear resistive grids, made from MOS transistors, have been found useful for image segmentation.[4,5] Networks of this type can be divided into two classes: feedback systems and feedforward-only systems. In the feedforward case one set of amplifiers imposes signal voltages or

currents on the grid and another set reads out the resulting response for subsequent processing, while the same amplifiers both "write" to the grid and "read" from it in a feedback arrangement. Feedforward networks of this type are inherently stable, but feedback networks need not be.

A practical example is one of Carver Mead's retina chips[3] that achieves edge enhancement by means of lateral inhibition through a resistive grid. Figure 1 shows a single cell in a continuous-time version of this chip. Note that the capacitor voltage is affected both by the local light intensity incident on that cell and by the capacitor voltages on neighboring cells of identical design. Any cell drives its neighbors, which drive both their distant neighbors and the original cell in turn. Thus the necessary ingredients for instability--active elements and signal feedback--are both present in this system, and in fact the continuous-time version oscillates so badly that the original design is scarcely usable in practice with the lateral inhibition paths enabled.[6] Such oscillations can

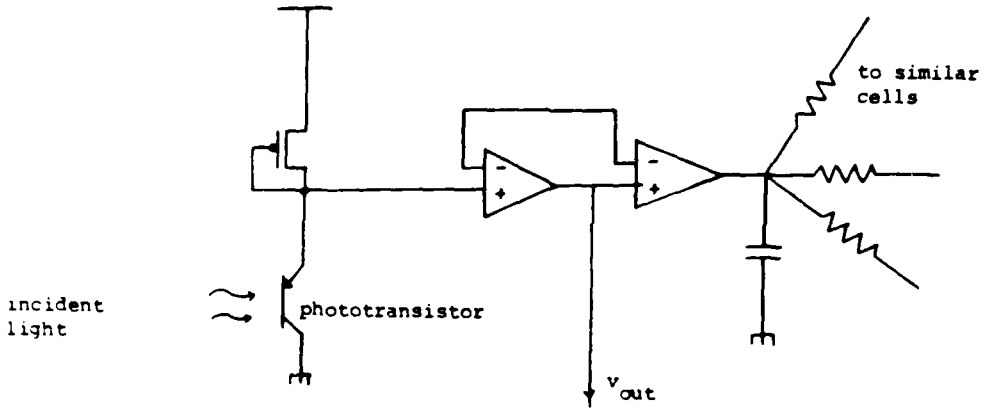



Figure 1. This photoreceptor and signal processor circuit, using two MOS transconductance amplifiers, realizes lateral inhibition by communicating with similar units through a resistive grid.

readily occur in any resistive grid circuit with active elements and feedback, even when each individual cell is quite stable. Analysis of the conditions of instability by straightforward methods appears hopeless, since any repeated array contains many cells, each of which influences many others directly or indirectly and is influenced by them in turn, so that the number of simultaneously active feedback loops is enormous.

This paper reports a practical design approach that rigorously guarantees such a system will be stable. The very simplest version of the idea is intuitively obvious: design each individual cell so that, although internally active, it acts like a passive system as seen from the resistive grid. In circuit theory language, the design goal here is that each cell's output impedance should be a *positive-real*[7] function. This is sometimes not too difficult in practice; we will show that the original network in Fig. 1 satisfies this condition in the absence of certain parasitic elements. More important, perhaps, it is a condition one can verify experimentally

by frequency-response measurements.

It is physically apparent that a collection of cells that appear passive at their terminals will form a stable system when interconnected through a passive medium such as a resistive grid. The research contributions, reported here in summary form, are i) a demonstration that this passivity or positive-real condition is much stronger than we actually need and that weaker conditions, more easily achieved in practice, suffice to guarantee stability of the linear network model, and ii) an extension of i) to the *nonlinear* domain that furthermore rules out large-signal oscillations under certain conditions.

## II. FIRST-ORDER LINEAR ANALYSIS OF A SINGLE CELL

We begin with a linear analysis of an elementary model for the circuit in Fig. 1. For an initial approximation to the output admittance of the cell we simplify the topology (without loss of relevant information) and use a naive model for the transconductance amplifiers, as shown in Fig. 2.

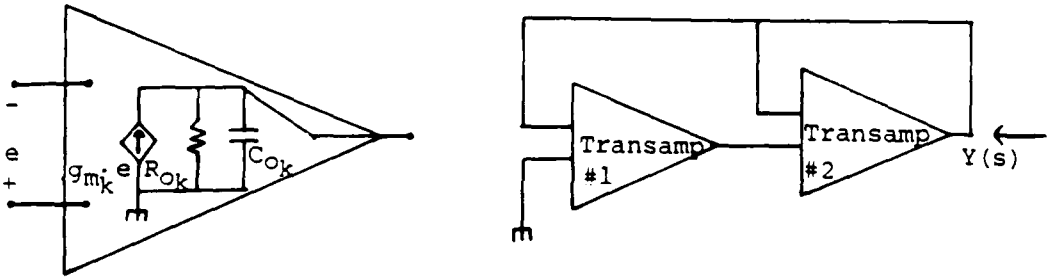



Figure 2. Simplified network topology and transconductance amplifier model for the circuit in Fig. 1. The capacitor in Fig. 1 has been absorbed into $C_{o_2}$.

Straightforward calculations show that the output admittance is given by

$$Y(s) = [g_{m_2} + R_{o_2}^{-1} + s\, C_{o_2}] + \frac{g_{m_1} g_{m_2} R_{o_1}}{(1 + s\, R_{o_1} C_{o_1})} \quad . \qquad (1)$$

This is a positive-real, i.e., passive, admittance since it can always be realized by a network of the form shown in Fig. 3, where $R_1 = (g_{m_2} + R_{o_2}^{-1})^{-1}$, $R_2 = (g_{m_1} g_{m_2} R_{o_1})^{-1}$, and $L = C_{o_1}/g_{m_1} g_{m_2}$.

Although the original circuit contains no inductors, the realization has both capacitors and inductors and thus is capable of damped oscillations. Nonetheless, *if* the transamp model in Fig. 2 were perfectly accurate, no network created by interconnecting such cells through a resistive grid (with parasitic capacitances) could exhibit sustained oscillations. For element values that may be typical in practice, the model in Fig. 3 has a lightly damped resonance around 1 KHz with a Q ≈ 10. This disturbingly high Q suggests that the cell will be highly sensitive to parasitic elements not captured by the simple models in Fig. 2. Our preliminary

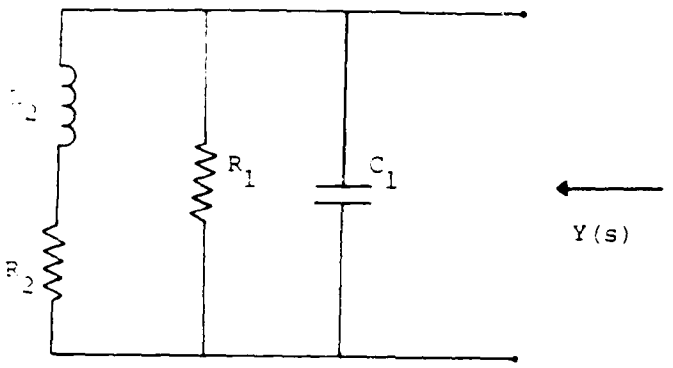



Figure 3. Passive network realization of the output admittance (eq. (1) of the circuit in Fig. 2.

analysis of a much more complex model extracted from a physical circuit layout created in Carver Mead's laboratory indicates that the output impedance will not be passive for all values of the trans-amp bias currents. But a definite explanation of the instability awaits a more careful circuit modelling effort and perhaps the design of an on-chip impedance measuring instrument.

## III. POSITIVE-REAL FUNCTIONS, θ-POSITIVE FUNCTIONS, AND STABILITY OF LINEAR NETWORK MODELS

In the following discussion $s = \sigma+j\omega$ is a complex variable, H(s) is a rational function (ratio of polynomials) in s with real coefficients, and we assume for simplicity that H(s) has no pure imaginary poles. The term *closed right half plane* refers to the set of complex numbers s with $\mathrm{Re}\{s\} \geq 0$.

Def. 1

The function H(s) is said to be *positive-real* if a) it has no poles in the right half plane and b) $\mathrm{Re}\{H(j\omega)\} \geq 0$ for all $\omega$.

If we know at the outset that H(s) has no right half plane poles, then Def. 1 reduces to a simple graphical criterion: H(s) is positive-real if and only if the *Nyquist diagram* of H(s) (i.e. the plot of $H(j\omega)$ for $\omega \geq 0$, as in Fig. 4) lies entirely in the closed right half plane.

Note that positive-real functions are necessarily stable since they have no right half plane poles, but stable functions are not necessarily positive-real, as Example 1 will show.

A deep link between positive real functions, physical networks and passivity is established by the classical result[7] in linear circuit theory which states that H(s) is positive-real if and only if it is possible to synthesize a 2-terminal network of positive linear resistors, capacitors, inductors and ideal transformers that has H(s) as its driving-point impedance or admittance.

Def. 2

The function H(s) is said to be θ-positive for a particular value of θ (θ ≠ 0, θ ≠ π), if a) H(s) has no poles in the right half plane, and b) the Nyquist plot of H(s) lies strictly to the right of the straight line passing through the origin at an angle θ to the real positive axis.

Note that every θ-positive function is stable and any function that is θ-positive with θ = π/2 is necessarily positive-real.

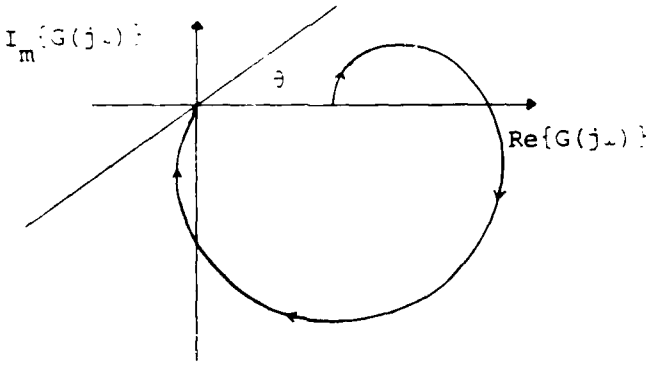



Figure 4. Nyquist diagram for a function that is θ-positive but not positive-real.

Example 1

The function

$$G(s) = \frac{(s+1)(s+40)}{(s+5)(s+6)(s+7)} \qquad (2)$$

is θ-positive (for any θ between about 18° and 68°) and stable, but it is not positive-real since its Nyquist diagram, shown in Fig. 4, crosses into the left half plane.

The importance of θ-positive functions lies in the following observations: 1) an interconnection of passive linear resistors and capacitors and cells with stable linear impedances can result in an unstable network, b) such an instability cannot result if the impedances are also positive-real, c) θ-positive impedances form a larger class than positive-real ones and hence θ-positivity is a less demanding synthesis goal, and d) Theorem 1 below shows that such an instability cannot result if the impedances are θ-positive, even if they are not positive-real.

Theorem 1

Consider a linear network of arbitrary topology, consisting of any number of passive 2-terminal resistors and capacitors of arbitrary value driven by any number of active cells. If the output impedances

of all the active cells are $\theta$-positive for some common $\theta$, $0<\theta\leq\frac{\pi}{2}$, then the network is stable.

The proof of Theorem 1 relies on Lemma 1 below.

Lemma 1

If H(s) is $\theta$-positive for some fixed $\theta$, then for all $s_o$ in the closed first quadrant of the complex plane, $H(s_o)$ lies strictly to the right of the straight line passing through the origin at an angle $\theta$ to the real positive axis, i.e., $\mathrm{Re}\{s_o\} \geq 0$ and $\mathrm{Im}\{s_o\} \geq 0 \Rightarrow$ $\theta - \pi < \angle H(s_o) < \theta$.

Proof of Lemma 1 (Outline)

Let d be the function that assigns to each s in the closed right half plane the perpendicular distance d(s) from H(s) to the line defined in Def. 2. Note that d(s) is harmonic in the closed right half plane, since H is analytic there. It then follows, by application of the maximum modulus principle[8] for harmonic functions, that d takes its minimum value on the boundary of its domain, which is the imaginary axis. This establishes Lemma 1.

Proof of Theorem 1 (Outline)

The network is unstable or marginally stable *if and only if* it has a natural frequency in the closed right half plane, and $s_o$ is a natural frequency if and only if the network equations have a nonzero solution at $s_o$. Let $\{I_k\}$ denote the complex branch currents of such a solution. By Tellegen's theorem[9] the sum of the complex powers absorbed by the circuit elements must vanish at such a solution, i.e.,

$$\sum_{\text{resistances}} R_k |I_k|^2 + \sum_{\text{capacitances}} |I_k|^2 / s_o C_k + \sum_{\substack{\text{cell} \\ \text{terminal pairs}}} Z_k(s_o) |I_k|^2 = 0, \tag{3}$$

where the second term is deleted in the special case $s_o=0$, since the complex power into capacitors vanishes at $s_o=0$.

If the network has a natural frequency in the closed right half plane, it must have one in the closed first quadrant since natural frequencies are either real or else occur in complex conjugate pairs. But (3) cannot be satisfied for any $s_o$ in the closed first quadrant, as we can see by dividing both sides of (3) by $\sum_k |I_k|^2$, where the sum is taken over all network branches. After this division, (3) asserts that zero is a convex combination of terms of the form $R_k$, terms of the form $(C_k s_o)^{-1}$, and terms of the form $Z_k(s_o)$. Visualize where these terms lie in the complex plane: the first set lies on the real positive axis, the second set lies in the closed 4-th quadrant since $s_o$ lies in the closed 1st quadrant by assumption, and the third set lies to the right of a line passing through the origin at an angle $\theta$ by Lemma 1. Thus all these terms lie strictly to the right of this line, which implies that no convex combination of them can equal zero. Hence the network is stable!

## IV. STABILITY RESULT FOR NETWORKS WITH NONLINEAR RESISTORS AND CAPACITORS

The previous result for linear networks can afford some limited insight into the behavior of nonlinear networks. First the nonlinear equations are linearized about an equilibrium point and Theorem 1 is applied to the linear model. If the linearized model is stable, then the equilibrium point of the original nonlinear network is *locally stable*, i.e., the network will return to that equilibrium point *if* the initial condition is sufficiently near it. But the result in this section, in contrast, applies to the full nonlinear circuit model and allows one to conclude that in certain circumstances the network cannot oscillate *even if* the initial state is *arbitrarily far from* the equilibrium point.

Def. 3

A function H(s) as described in Section III is said to satisfy the *Popov criterion*[10] if there exists a real number r>0 such that $\mathrm{Re}\{(1+j\omega r)\ H(j\omega)\} \geq 0$ for all $\omega$.

Note that positive real functions satisfy the Popov criterion with r=0. And the reader can easily verify that G(s) in Example 1 satisfies the Popov criterion for a range of values of r. The important effect of the term $(1+j\omega r)$ in Def. 3 is to rotate the Nyquist plot counterclockwise by progressively greater amounts up to 90° as $\omega$ increases.

Theorem 2

Consider a network consisting of nonlinear 2-terminal resistors and capacitors, and cells with linear output impedances $Z_k(s)$. Suppose

i) the resistor curves are characterized by continuously differentiable functions $i_k = g_k(v_k)$ where $g_k(0) = 0$ and $0 < g_k'(v_k) < G < \infty$ for all values of k and $v_k$,

ii) the capacitors are characterized by $i_k = C_k(v_k)\dot{v}_k$ with $0 < C_1 < C_k(v_k) < C_2 < \infty$ for all values of k and $\dot{v}_k$,

iii) the impedances $Z_k(s)$ have no poles in the closed right half plane and all satisfy the Popov criterion for some common value of r.

If these conditions are satisfied, then the network is stable in the sense that, for any initial condition,

$$\int_0^\infty \left[ \sum_{\text{all branches}} i_k^2(t) \right] dt < \infty \ . \qquad (4)$$

The proof, based on Tellegen's theorem, is rather involved. It will be omitted here and will appear elsewhere.

ACKNOWLEDGEMENT

We sincerely thank Professor Carver Mead of Cal Tech for enthusiastically supporting this work and for making it possible for us to present an early report on it in this conference proceedings. This work was supported by Defense Advanced Research Projects Agency (DoD), through the Office of Naval Research under ARPA Order No. 3872, Contract No. N00014-80-C-0622 and Defense Advanced Research Projects Agency (DARPA) Contract No. N00014-87-K-0825.

REFERENCES

1. F.S. Werblin, "The Control of Sensitivity on the Retina," Scientific American, Vol. 228, no. 1, Jan. 1983, pp. 70-79.
2. T. Kohonen, Self-Organization and Associative Memory, (vol. 8 in the Springer Series in Information Sciences), Springer Verlag, New York, 1984.
3. M.A. Sivilotti, M.A. Mahowald, and C.A. Mead, "Real Time Visual Computations Using Analog CMOS Processing Arrays," Advanced Research in VLSI - Proceedings of the 1987 Stanford Conference, P. Losleben, ed., MIT Press, 1987, pp. 295-312.
4. C.A. Mead, Analog VLSI and Neural Systems, Addison-Wesley, to appear in 1988.
5. J. Hutchinson, C. Koch, J. Luo and C. Mead, "Computing Motion Using Analog and Binary Resistive Networks," submitted to IEEE Transactions on Computers, August 1987.
6. M. Mahowald, personal communication.
7. B.D.O. Anderson and S. Vongpanitlerd, Network Analysis and Synthesis - A Modern Systems Theory Approach, Prentice-Hall, Englewood Cliffs, NJ., 1973.
8. L.V. Ahlfors, Complex Analysis, McGraw-Hill, New York, 1966, p. 164.
9. P. Penfield, Jr., R. Spence, and S. Duinker, Tellegen's Theorem and Electrical Networks, MIT Press, Cambridge, MA, 1970.
10. M. Vidyasagar, Nonlinear Systems Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1970, pp. 211-217.



VLSI Memo No. 87-422
November 1987

# CONCURRENT COMPUTER ARCHITECTURE

William J. Dally

## Abstract

Present generation concurrent computers offer performance greater than vector supercomputers and are easily programmed by non-experts. Evolution of VLSI technology and a better understanding of concurrent machine organization have led to substantial improvements in the performance of numerical processors, symbolic processors, and communication networks. A 100MFLOPS arithmetic chip and a 5$\mu$s latency communication network are under construction. Low-latency communication and task switching simplify concurrent programming by removing considerations of grain size and locality. A message-passing concurrent computer with a global virtual address space provides programmers with both a shared memory, and message-based communication and synchronization. This paper describes recent advances in concurrent computer architecture drawing on examples from the J-Machine, and experimental concurrent computer under development at MIT.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138

# 18.435/6.848 Theory of Parallel and VLSI Computation

## Lecture Notes


Lecturers: Tom Leighton
Charles E. Leiserson

Teaching Assistants: Bruce Maggs
Serge Plotkin
Joel Wein



Fall 1987

# Preface

This manuscript consists of the lecture notes for *6.848/18.435 Theory of Parallel and VLSI Computation* as it was taught by Professors Leighton and Leiserson in the Fall of 1987. The notes were written by students in the class and were reviewed and organized by Bruce Maggs, Serge Plotkin, and Joel Wein, who served as Teaching Assistants for the class. The notes are not meant to be in polished form, and they probably contain several errors and omissions, particularly with respect to references in the literature. Rather, they are intended to be an aid in teaching and/or studying the introductory work in networks, parallel computation, and VLSI Design. For a more complete and formal treatment of the work in this area, we refer the reader to the references listed in the reading assignments and/or Professor Leighton's forthcoming book on this subject.

The class was attended by about 35 students, 25 of whom took the course for credit. No previous familiarity with networks, parallel computation, or VLSI was assumed, but we did require that students have gained some familiarity with data structures and algorithms before taking the course. The lectures were supplemented with biweekly problem sets and a list of topics for research.

It is our intention to update these notes each year that the course is taught, and we would appreciate knowing of any errors and omissions in the current notes so that they may be corrected in future notes. Currently, we plan to teach the course during each fall term.

Notes for the advanced course on this subject (*6.849/18.436 Advanced Parallel and VLSI Computation*) are also available as an MIT-LCS Research Seminar Series RSS-2. The advanced course was last taught in the Spring of 1987, and we plan to teach it every other spring term henceforth.

## List of Lectures

# 18.436/6.849 Advanced Parallel and VLSI Computation

## Lecture Notes


| | |
|---|---|
| Lecturers: | Tom Leighton<br>Charles E. Leiserson |
| Teaching Assistants: | Bruce Maggs<br>Serge Plotkin<br>Joel Wein |



Spring 1987

# Preface

This manuscript consists of the lecture notes for *6.849/18.436 Advanced Parallel and VLSI Computation* as it was taught by Professors Leighton and Leiserson in the Spring of 1987. The notes were written by students in the class and were reviewed and organized by Bruce Maggs, Serge Plotkin, and Joel Wein, who served as Teaching Assistants for the class. The notes are not meant to be in polished form, and they probably contain several errors and omissions, particularly with respect to references in the literature. Rather, they are intended to be an aid in teaching and for studying some of the advanced work in networks, parallel computation, and VLSI design. For a more complete and formal treatment of the work in this area, we refer the reader to the references listed in the reading assignments and/or Professor Leighton's forthcoming book on this subject.

The class was attended by about 25 students, 15 of whom took the course for credit. Students were assumed to have taken an introductory course on this subject (e.g. *6.848/18.435 Theory of Parallel and VLSI Computation*) as a prerequisite. The lectures were supplemented with monthly problem sets and two programming projects on a 16,000-processor Connection Machine provided by Thinking Machines Corporation of Cambridge, Mass.

It is our intention to update these notes each year that the course is taught. Currently, we plan to teach the course every other spring. We would appreciate learning of any errors and omissions in the current notes so that they can be corrected in future editions.

Notes for the introductory course on this material (*6.848/18.435 Theory of Parallel and VLSI Computation*) are also available as an MIT-LCS Research Seminar Series RSS-1. The introductory course was last taught during the Fall of 1987, and is scheduled to be taught during each Fall Term henceforth.

## List of Lectures



VLSI Memo No. 88-444
March 1988

# A MIXED FREQUENCY-TIME APPROACH FOR FINDING THE STEADY-STATE SOLUTION OF CLOCKED ANALOG CIRCUITS

K. Kundert, J. White, and A. Sangiovanni-Vincentelli

Abstract

Performing detailed simulation of clocked analog circuits (e.g. switched-capacitor filters and switching power supplies) with circuit simulation programs like SPICE is computationally very expensive. In this paper we present a new, more efficient, method for computing the detailed steady-state solution of clocked analog circuits. The method exploits the property of such circuits that the waveforms in each clock cycle are similar but not exact duplicates of the proceeding or following cycles. Therefore, by computing accurately a few selected cycles, the entire steady-state solution can be constructed efficiently.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138



VLSI Memo No. 88-436
February 1988

# FAST ON-CHIP DELAY ESTIMATION FOR CELL-BASED EMITTER COUPLED LOGIC

Peter R. O'Brien, John L. Wyatt, Jr., Thomas L. Savarino, and James M. Pierce

Abstract

The goal of this work is to produce fast, but accurate, estimates of best and worst case delay for on-chip emitter coupled logic (ECL) nets. The work consists of two major parts: 1) macromodelling of ECL logic gates acting as both sources and loads; and 2) delay estimation for individual nets using the gate macromodel parameters and RC tree models for metal interconnect. Both of the above functions (gate macromodeling and delay estimation) have been extensively tested on an industrial ECL process and cell (i.e., logic gate) library.

The success of a macromodelling approach relies on repetitive use of members of a library of modelled cells. A "fixed" computational cost (several c.p.u. hours per cell) is paid to obtain simplified macromodel parameter values. Resultant timing estimates are typically within 5-10% of SPICE and are obtained roughly three orders of magnitude more quickly than SPICE.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

# FAST ON-CHIP DELAY ESTIMATION FOR CELL-BASED EMITTER COUPLED LOGIC

Peter R. O'Brien, John L. Wyatt, Jr., Thomas L. Savarino, James M. Pierce

O'Brien, Pierce: Digital Equipment Corporation, Marlborough, MA 01752
Wyatt: Massachusetts Institute of Technology, Cambridge, MA 02139
Savarino: Tangent Systems Corporation, Santa Clara, CA 95051

## ABSTRACT

The goal of this work is to produce fast, but accurate, estimates of best and worst case delay for on-chip emitter coupled logic (ECL) nets. The work consists of two major parts: 1) macromodelling of ECL logic gates acting as both sources and loads; and 2) delay estimation for individual nets using the gate macromodel parameters and RC tree models for metal interconnect. Both of the above functions (gate macromodelling and delay estimation) have been extensively tested on an industrial ECL process and cell (i.e., logic gate) library.

The success of a macromodelling approach relies on repetitive use of members of a library of modelled cells. A "fixed" computational cost (several c.p.u. hours per cell) is paid to obtain simplified macromodel parameter values. Resultant timing estimates are typically within 5-10% of SPICE 1 and are obtained roughly three orders of magnitude more quickly than SPICE.

## I. INTRODUCTION

### Definition of terms

The definition of "metal delay" can best be illustrated by a simplified interconnect net with no branching and only one load gate [Fig. 1]. Let $T_{AB}(0)$ represent delay through the *unloaded* source gate. Let $T_{AB}(L)$ represent delay through the source gate *loaded* by an interconnect net of length $L$, as shown in Fig. 1. For all gates in our cell library, $T_{AB}(0)$ is known. What

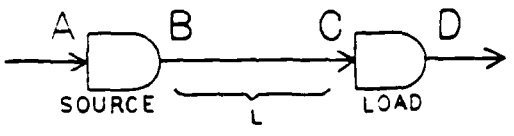



Figure 1: Simplified interconnect net.

our algorithm estimates is "metal delay" defined by:

$$D_{metal} \triangleq T_{AC} - T_{AB}(0) = T_{AB}(L) - T_{AB}(0) + T_{BC} \quad (1)$$

So, "metal delay" has two distinct components: *extra delay through the source gate* caused by the loading of the source gate by metal, and *propagation delay* through the metal itself. Worst case (or "slow") metal delay is simply the definition in (1) evaluated using slow SPICE process parameters for the logic gates and metal interconnect, the maximum expected input risetime at point $A$, and a slow target voltage threshold at points $B$ and $C$. The slow target voltage thresholds for rising and falling transitions are chosen to be, respectively:

$$V_{T,slow\ rise} \triangleq \left(\frac{V_{LOW} + V_{HIGH}}{2}\right) + V_{noise\ margin} \quad (2)$$

$$V_{T,slow\ fall} \triangleq \left(\frac{V_{LOW} + V_{HIGH}}{2}\right) - V_{noise\ margin}, \quad (3)$$

for some $V_{noise\ margin} > 0$. The definition of best case (or "fast") metal delay is made in a similar way with fast versions of SPICE parameters, input risetime, and output voltage threshold.

### Gate delay vs. interconnect delay

Previous work on waveform bounding and estimation for RC tree networks [2,3,4], with application to MOS circuits, has focused on the propagation component ($T_{BC}$) of interconnect delay. Relatively simple models are used for the logic gates themselves. More recently, detailed macromodels for MOS logic gates [5] have been developed and used in conjunction with the RC tree delay estimation results. Good correlation with SPICE is obtained by fitting macromodel parameters to selected SPICE experiments. We develop macromodels specifically for ECL gates at a level of detail similar to [5]. This fills a definite need since most reported work in this area has been for MOS circuits, even though bipolar digital circuits are presently in wide use for high-performance applications. Recent work that has been reported for bipolar circuits [6,7,8] is concerned mainly with logic simulation, and the timing models used are relatively simple.

To emphasize the importance of accurately modelling the source gate (as opposed to just the interconnect), in Fig. 2 we plot separately the two components of (rising transition, worst case) metal delay versus total load net capacitance for a *uniformly distributed* metal line in the simplified topology of Fig. 1. The gate used as both the source and the load is a stan-

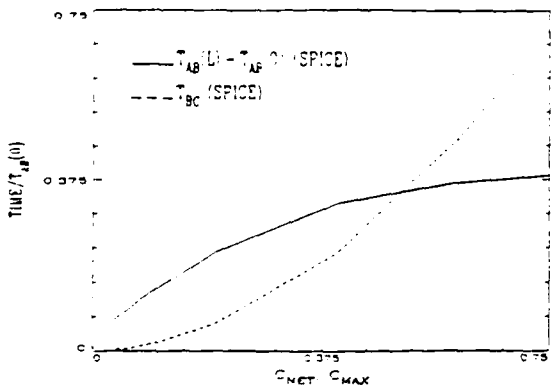



Figure 2: Two components of $D_{metal}$ vs. total load net capacitance.

dard 2-input OR NOR. We denote the total load net capacitance by "$C_{NET}$," and the maximum expected value of $C_{NET}$ by "$C_{MAX}$." For low values of $C_{NET}$, extra source gate delay dominates propagation delay; the two become equal only when $C_{NET}/C_{MAX} \approx 0.51$. Furthermore, the statistical distribution of $C_{NET}$ is not uniform across $[0, C_{MAX}]$, but rather is skewed towards *lower* capacitance values. In fact, for our designs, 90% of the load nets have $C_{NET}/C_{MAX}$ values under 0.25, where propagation delay is only 42% of extra source gate delay. In addition, for a falling transition, extra source gate delay is even more important than shown in Fig. 2, typically exceeding propagation delay throughout the *entire* range $0 \leq C_{NET} \leq C_{MAX}$. Therefore, extra source gate delay is typically the dominant component of metal delay.

## II. LOGIC GATE MACROMODELLING

### Load modelling

Modelling of ECL gates as *loads* is very simple. As in MOS, a single linear lumped capacitor is an acceptable load model. Modelling of leakage current in ECL loads does not appear to be necessary. For our process, the worst case (maximum expected metal resistance, maximum expected fanout) voltage drop through metal due to steady state leakage current is less than 3% of the rail-to-rail voltage swing. In situations where leakage current might be modelled (e.g., *clock* nets with very high fanout), this could be done by adding a linear resistor and a d.c. current source in parallel with the capacitor [9,10]. The capacitance value is extracted from SPICE simulations of transient voltage at the load and current into the load. Since on-chip metal is modelled as a linear RC tree, and load gates as simple linear capacitors, this means an *entire* load net (metal and loads) is modelled as a linear RC tree.

### Source modelling

Modelling of ECL gates as *sources* is more difficult. Because of an emitter-follower output stage, source gate behavior exhibits a fundamental asymmetry between rising and falling transitions. For a falling transition, there is a limitation on transient sinking current as $C_{NET}$ increases. We use two different source model types: the first one for falling transitions when $C_{NET} \geq C_{THRESH}$ (to model the transient sinking current limitation), and the second one for falling transitions when $C_{NET} < C_{THRESH}$ and for all rising transitions. The "threshold" capacitance ($C_{THRESH}$) is determined from SPICE simulations of transient source gate output current ($I_{out}$) during falling transitions. $C_{THRESH}$ is defined to be the value of $C_{NET}$ where the *sensitivity* of $|I_{out}|_{max}$ to a perturbation in $C_{NET}$ drops below a predetermined value.

To model the sinking current limitation, the first source gate model is simply a delayed current source *pulse*. The model delay before the onset of the current pulse and the magnitude of the pulse ($I_{SAT}$) are extracted from the same SPICE simulations used to determine $C_{THRESH}$. The duration of the pulse is exactly long enough to sink the correct amount of charge:

$$\text{pulse duration} \triangleq \frac{C_{NET}(V_{HIGH} - V_{LOW})}{I_{SAT}}. \qquad (4)$$

The second source gate model is based on the source gate's *d.c. drive curves*, which show the static output voltage vs. input voltage relationship for different values of output load current. To describe a family of three-segment piecewise-linear approximations to the d.c. drive curves, four "d.c. parameters" are obtained by curve fitting to d.c. SPICE output (see also [3,5]). These d.c. parameters alone are sufficient to model the source gate's response to *slow* inputs, when the gate behaves quasi-statically. However, an ECL input is usually too fast for the source gate to respond quasi-statically. The source gate responds somewhat more slowly than the quasi-static model alone would predict. So, four additional "dynamic parameters" are extracted from SPICE data of *transient* source gate responses in order to model, as a function of $C_{NET}$, the departure from a purely quasi-static response.

Each of the two source gate models is used in conjunction with an approximation to the driving-point

admittance of the load net given by a single lumped RC segment ($R_{NET}$, $C_{NET}$). Based on values for $R_{NET}$, $C_{NET}$, the source gate macromodel parameters, and the input risetime, a closed-form *analytical* expression for the model waveform, $v_B(t)$ is obtained. The detailed model expressions are omitted here for brevity, but can be found in [11].

Macromodel parameter summary

A total of $2(lev)$ load gate macromodel parameter values (capacitance) are extracted for each cell, where $lev$ denotes the number of input levels of the cell being modelled. (Note: the term "input level" refers to a subset of the individual input signals that affect the current-steering logic through the same number of base-emitter junction voltage drops.) Capacitance values are obtained for each combination of slow/fast SPICE parameters and different input level.

A total of 76 source gate macromodel parameter values are extracted for each cell: 12 for the first source model (i.e., falling transition *and* $C_{NET} \geq C_{THRESH}$), and 64 for the second source model. For the first model: 3 parameters ($C_{THRESH}$, $I_{SAT}$, and current pulse delay) for each combination of slow/fast SPICE parameters and "true/complement" output side of the cell. For the second model: 8 parameters (4 "d.c." and 4 "dynamic") for each combination of slow/fast SPICE parameters, "true/complement" output side of the cell, and rising/falling output transition.

## III. REDUCED-ORDER INTERCONNECT MODELS

Driving-point admittance approximation

As mentioned in the previous section, to enable the computation of an analytical expression for the model waveform $v_B(t)$, the driving-point admittance of the load net is approximated by a single lumped RC segment ($R_{NET}$,$C_{NET}$). The values of $R_{NET}$ and $C_{NET}$ are chosen to match the first two terms of the Taylor series expansion around $s = 0$ of the driving-point admittance of the given load net,

$$Y_{LOAD\,NET}(s) = \sum_{n=1}^{\infty} y_n s^n, \qquad (5)$$

where the series representation is valid only within some circle of convergence $|s| < r_{conv}$. Our approximate driving-point admittance is:

$$\begin{aligned} Y_{APPROX}(s) &= \frac{sC_{NET}}{1 - sR_{NET}C_{NET}} \\ &= \sum_{n=1}^{\infty} (-1)^{n-1} {R_{NET}}^{n-1} {C_{NET}}^{n} s^n, \end{aligned} \qquad (6)$$

where the second equality is valid only within the circle of convergence $|s| < 1/R_{NET}C_{NET}$. So to match both the $s$ and $s^2$ terms, we choose:

$$C_{NET} = y_1 \qquad (7)$$

$$R_{NET} = -y_2 / y_1^2 \qquad (8)$$

The approximation is computed quickly using an algorithm [11] which allows the computation of $y_1$ and $y_2$ (and, hence, of $R_{NET}$ and $C_{NET}$) to proceed *sequentially upstream* from the leaf nodes of the load net until the source gate output is finally reached. The *low-frequency* nature of this approximation, implicit in the use of a Taylor series expansion around $s = 0$, turns out to be entirely justified. For our process, most of the frequency content of a typical source gate output waveform lies well inside the circle of convergence for both the true and approximate driving-point admittance [11].

Voltage transfer function approximation

We propagate the model source gate output voltage waveform ($v_B(t)$) downstream to the load(s) of interest by *convolving* with an approximate unit voltage impulse response first developed by Horowitz [3]. We use an approximate impulse response because obtaining the precise impulse response of a general RC tree is too *computationally expensive*. In addition, closed-form analytical expressions are available for the approximate impulse response. This allows, via convolution with the model source gate output voltage waveform, computation of closed-form analytical expressions for the model voltage waveform(s) at the load(s) ($v_C(t)$). The model voltage waveform at each load is then numerically inverted, at the appropriate voltage threshold, in order to compute the metal delay to that load.

Let $h(t)$ and $h_{approx}(t)$ denote, respectively, the true and approximate unit voltage impulse response at a given load. Let $H(s)$ and $H_{approx}(s)$ denote, respectively at the same load, the Laplace transform of the true and approximate impulse response. The approximate transfer function has *two poles and one zero*:

$$H_{approx}(s) = \frac{1 + s\tau_z}{(1 + s\tau_1)(1 + s\tau_2)}. \qquad (9)$$

The time constants $\tau_z$, $\tau_1$, and $\tau_2$ are determined by the following three constraints:

$$\int_0^{\infty} t h_{approx}(t)dt = \int_0^{\infty} t h(t)dt \qquad (10)$$

$$\int_0^{\infty} t^2 h_{approx}(t)dt = \int_0^{\infty} t^2 h(t)dt \qquad (11)$$

$$\frac{1 + b_1 s + b_2 s^2 + \cdots}{1 + (\tau_1 + \tau_2)\, s + a_2 s^2 + \cdots} = H(s). \qquad (12)$$

## IV. RESULTS

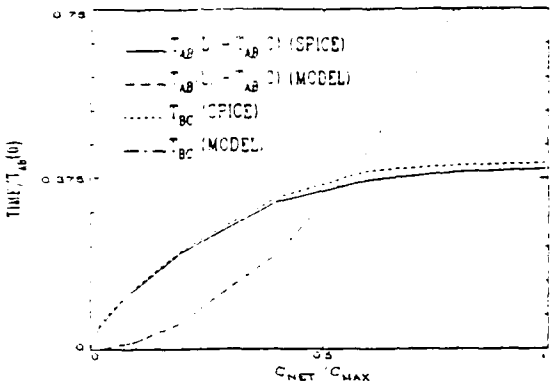



Figure 3: Rising transition comparison.

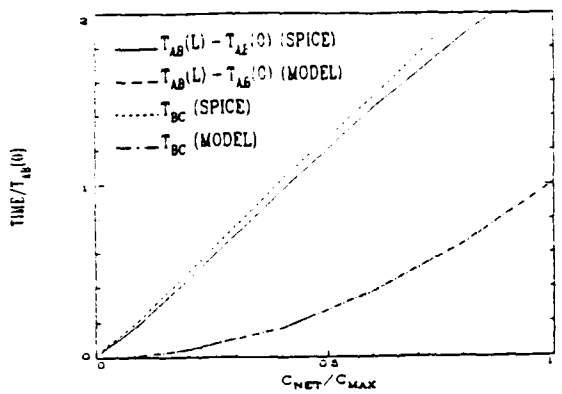



Figure 4: Falling transition comparison.

In Figs. 3 and 4, we show comparisons of SPICE vs. our algorithm. In Fig. 3, we use the same SPICE data shown in Fig. 2. In Fig. 4, we use the same gate (2-input OR/NOR) and the same net topology (uniform unbranched line with a single load), but we examine a *falling* transition. Two points to note about Fig. 4 are:

1. the boundary between the two different source model types is $C_{NET} = C_{THRESH} = 0.43C_{MAX}$ for this particular gate; and

2. the two $T_{BC}$ curves are nearly indistinguishable on the time scale of the plot.

Assuming that gate macromodel parameters have been obtained in advance, the computation speed-up relative to SPICE is approximately three orders of magnitude. Similar accuracy and speed-up results are obtained using a wide variety of different logic cells and non-uniform branched net topologies [11].

## ACKNOWLEDGMENTS

This work was supported by the Digital Equipment Corporation, the National Science Foundation under Grant No. ECS83-10941, and the Defense Advanced Research Projects Agency under Contracts No. N00014-80-C-0622 and N00014-87-K-0825.

## REFERENCES

[1] L.W. Nagel, *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, Memo ERL-M520, University of California, Berkeley, May 1975.

[2] J. Rubinstein, P. Penfield, Jr., and M.A. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 3, pp. 202-211, July 1983.

[3] M.A. Horowitz, *Timing Models for MOS Circuits*, Ph.D. Thesis, Stanford University, 1983.

[4] T. Lin and C.A. Mead, "Signal Delay in General RC Networks," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, no. 4, pp. 331-349, October 1984.

[5] M.D. Matson, *Macromodelling and Optimization of Digital MOS VLSI Circuits*, Ph.D. Thesis, Massachusetts Institute of Technology, 1985.

[6] P. Kozak, A.K. Bose, and A. Gupta, "Design Aids for the Simulation of Bipolar Gate Arrays," *Proc. 20th Design Automation Conference*, Miami Beach, FL, pp. 286-292, June 1983.

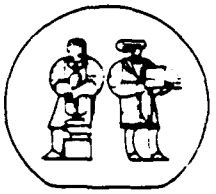
[7] I.N. Hajj, D. Saab, and B. Rosario, "Logic and Timing Simulation of Bipolar ECL Circuits," *Proc. 1984 IEEE Int. Conference on Computer-Aided Design*, Santa Clara, CA, pp. 194-196, November 1984.

[8] I.N. Hajj and D. Saab, "Switch-Level Logic Simulation of Digital Bipolar Circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 2, pp. 251-258, March 1987.

[9] P. O'Brien and J.L. Wyatt, Jr., *Signal Delay in Leaky RC Mesh Models for Bipolar Interconnect*, M.I.T. VLSI Memo 85-278, November 1985. All VLSI Memos are available from the Microsystems Research Center, Room 39-321, M.I.T., Cambridge, MA 02139.

[10] P. O'Brien and J.L. Wyatt, Jr., "Signal Delay in ECL Interconnect," *Proc. 1986 IEEE Int. Symp. on Circuits and Systems*, pp. 755-758, San Jose, CA, May 1986.

[11] P. O'Brien and J.L. Wyatt, Jr., *Fast On-Chip Delay Estimation for Cell-Based Emitter Coupled Logic*, To appear in M.I.T. VLSI Memo Series in 1988. All VLSI Memos are available from the Microsystems Research Center, Room 39-321, M.I.T., Cambridge, MA 02139.





VLSI Memo No. 88-439
March 1988

# CIRCUIT DESIGN CRITERIA FOR STABLE LATERAL INHIBITION NEURAL NETWORKS

J. L. Wyatt, Jr. and D. L. Standley

Abstract

In the analog VLSI implementation of neural systems, it is sometimes convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. A serious problem of unwanted spontaneous oscillation often arises with these circuits and renders them unusable in practice. This paper reports a design approach that guarantees such a system will be stable, even though the values of designed elements in the resistive grid may be imprecise and the location and values of parasitic elements may be unknown. The method is based on a rigorous, somewhat novel mathematical analysis using Tellegen's theorem and the idea of Popov multipliers from control theory. It is thoroughly practical because the criteria are **local** in the sense that no overall analysis of the interconnected system is required, **empirical** in the sense that they involve only measurable frequency response data on the individual cells, and **robust** in the sense that unmodelled parasitic resistances and capacitances in the interconnect network cannot affect the analysis.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

# CIRCUIT DESIGN CRITERIA FOR STABLE LATERAL INHIBITION NEURAL NETWORKS

J.L. WYATT, Jr. and D.L. STANDLEY

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

## ABSTRACT

In the analog VLSI implementation of neural systems, it is sometimes convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. A serious problem of unwanted spontaneous oscillation often arises with these circuits and renders them unusable in practice. This paper reports a design approach that guarantees such a system will be stable, even though the values of designed elements in the resistive grid may be imprecise and the location and values of parasitic elements may be unknown. The method is based on a rigorous, somewhat novel mathematical analysis using Tellegen's theorem and the idea of Popov multipliers from control theory. It is thoroughly practical because the criteria are local in the sense that no overall analysis of the interconnected system is required, empirical in the sense that they involve only measurable frequency response data on the individual cells, and robust in the sense that unmodelled parasitic resistances and capacitances in the interconnect network cannot affect the analysis.

## I. INTRODUCTION

The term "lateral inhibition" first arose in neurophysiology to describe a common form of neural circuitry in which the output of each neuron in some population is used to inhibit the response of each of its neighbors. Perhaps the best understood example is the horizontal cell layer in the vertebrate retina, in which lateral inhibition simultaneously enhances intensity edges and acts as an automatic gain control to extend the dynamic range of the retina as a whole [1]. The principle has been used in the design of artificial neural system algorithms by Kohonen [2] and others and in the electronic design of neural chips by Carver Mead et. al. [3,4].

In the VLSI implementation of neural systems, it is convenient to build lateral inhibition networks by using a locally connected on-chip resistive grid. Linear resistors fabricated in, e.g., polysilicon, yield a very compact realization, and nonlinear resistive grids, made from MOS transistors, have been found useful for image segmentation. [4, 5]. Networks of this type can be divided into two classes: feedback systems and feedforward-only systems. In the feedforward case one set of amplifiers imposes signal voltages or currents on the grid and another set reads out the resulting response for subsequent processing, while the same amplifiers both "write" to the grid and "read" from it in a feedback arrangement. Feedforward networks of this type are inherently stable, but feedback networks need not be.

A practical example is one of Carver Mead's retina chips [3] that achieves edge enhancement by means of lateral inhibition through a resistive grid. Figure 1 shows a single cell in a continuous-time version of this chip. Note that the capacitor voltage is affected both by the local light intensity incident on that cell and by the capacitor voltages on neighboring cells of identical design. Any cell drives its neighbors, which drive both their distant neighbors and the original cell in turn. Thus the necessary ingredients for instability--active elements and signal feedback--are both present in this system, and in fact the continuous-time version oscillates so badly that the original design is scarcely usable in practice with the lateral inhibition paths enabled. [6] Such oscillations can

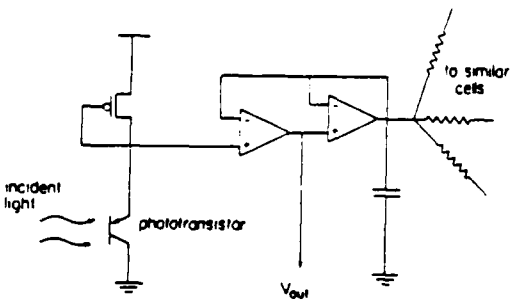



Figure 1. This photoreceptor and signal processor circuit, using two MOS transconductance amplifiers, realizes lateral inhibition by communicating with similar units through a resistive grid.

readily occur in any resistive grid circuit with active elements and feedback, even when each individual cell is quite stable. Analysis of the conditions of instability by straightforward methods appears hopeless, since the number of simultaneously active feedback loops is enormous.

This paper reports a practical design approach that rigorously guarantees such a system will be stable. The very simplest version of the idea is intuitively obvious: design each individual cell so that, although internally active, it acts like a passive system as seen from the resistive grid. In

circuit theory language, the design goal here is that each cell's output impedance should be a *positive-real* [7] function. This is sometimes not too difficult in practice; we will show that the original network in Fig. 1 satisfies this condition in the absence of certain parasitic elements. More important, perhaps, it is a condition one can verify experimentally by frequency-response measurements.

It is physically apparent that a collection of cells that appear passive at their terminals will form a stable system when interconnected through a passive medium such as a resistive grid. The research contributions, reported here in summary form, are i) a demonstration that this passivity or positive-real condition is much stronger than we actually need and that weaker conditions, more easily achieved in practice, suffice to guarantee stability of the linear network model, and ii) an extension to the *nonlinear* domain that furthermore rules out large-scale oscillations under certain conditions.

## II. FIRST-ORDER LINEAR ANALYSIS OF A SINGLE CELL

We begin with a linear analysis of an elementary model for the circuit in Fig. 1. For an initial approximation to the output admittance of the cell we simplify the topology (without loss of relevant information) and use a naive' model for the transconductance amplifiers, as shown in Fig. 2.

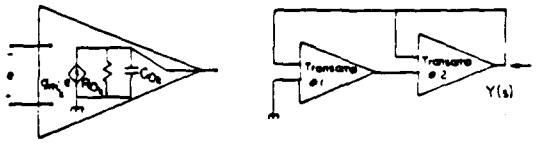



Figure 2. Simplified network topology and transconductance amplifier model for the circuit in Fig. 1. The capacitor in Fig. 1 has been absorbed into $C_{o_2}$.

Straightforward calculations show that the output admittance is

$$Y(s) = [g_{m_2}+R_{o_2}^{-1}+s\,C_{o_2}] + \frac{g_{m_1}g_{m_2}R_{o_1}}{(1+s\,R_{o_1}C_{o_1})} \quad . \quad (1)$$

This is a positive-real, i.e., passive, admittance that could always be realized by a network of the form shown in Fig. 3, where

$$R_1=(g_{m_2}+R_{o_1}^{-1})^{-1},\ R_2=(g_{m_1}g_{m_2}R_{o_1})^{-1},\ \text{and}\ L = C_{o_1}/g_{m_1}g_{m_2}.$$

Although the original circuit contains no inductors, the realization has both capacitors and inductors and thus is capable of damped oscillations. Nonetheless, *if* the transamp model in Fig. 2 were perfectly accurate, no network created by interconnecting such cells through a resistive grid (with parasitic capacitances) could exhibit sustained oscillations since all the elements are passive. For element values that may be typical in practice, the model in Fig. 3 has a lightly damped resonance around 1 KHz with a Q = 10. This disturbingly high Q suggests that the cell will be highly sensitive to parasitic elements not captured by the simple models in Fig. 2. Our preliminary analysis of a

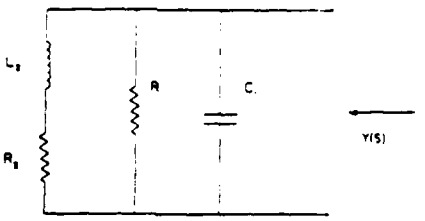



Figure 3. Passive network realization of the output admittance (eq. (1)) of the circuit in Fig. 1.

much more complex model extracted from a physical circuit layout created in Carver Mead's laboratory indicated that the output impedance will not be passive for all values of the transamp bias currents. But a definite explanation of the instability awaits a more careful circuit modelling effort and perhaps the design of an on-chip impedance measuring instrument.

## III. STABILITY OF A LINEAR MODEL FOR THE NETWORK

Transistor parasitics and layout parasitics will cause the output admittance of the individual active cells to deviate from the form given in eq. (1) and Fig. 3, and any very accurate model will necessarily be quite high order. The following theorem shows what sort of deviations we can allow and still guarantee that the network is stable.

### Terminology

The term *closed right half plane* refers to the set of complex numbers $s = \sigma + j\omega$ with $\sigma \geq 0$ and the term *closed third quadrant* refers to the set of complex numbers with $\sigma \leq 0$ and $\omega \leq 0$. A *natural frequency* is a complex frequency $s_o$ such that, when all branch impedances and admittances are evaluated at $s_o$, there exists a nonzero solution for the complex branch voltages $\{V_k\}$ and currents $\{I_k\}$.

### Theorem 1

Consider a linear network of arbitrary topology, consisting of any number of positive 2-terminal resistors and capacitors and of N lumped linear admittances $Y_n(s)$, $n=1,2,\ldots,N$, having no poles or zeroes in the closed right half plane. Then the network is *stable*, in the sense that it has no natural frequency in the closed right half plane except perhaps at the origin, if at each frequency $\omega \geq 0$ there exists a phase angle $\theta(\omega)$ such that $0 \leq \theta(\omega) \leq 90°$ and $|\angle Y_n(j\omega) - \theta(j\omega)| < 90°$, $n=1,2,\ldots,N$.

An equivalent statement of this last condition is that the Nyquist plot of each cell's output admittance for $\omega \geq 0$ never intersects the closed 3rd quadrant, and that no two cell's output admittance phase angles can ever differ by as much as 180°. If all the active cells are designed identically and fabricated on the same chip, their phase angles

should track closely in practice and thus this second condition is a natural one.

Note that the above statement of the theorem does not rule out the possibility of an unusual instability arising from a repeated natural frequency at the origin. But a more careful argument, omitted here, shows that the only possible nonzero network solution at s=0 is the stable one in which capacitors in capacitor-only loops have nonzero d.c. voltages and all other branch voltages and currents vanish.

Proof of Theorem 1

Let $s_0$ denote a natural frequency of the network and $\{V_k\}$ denote the complex branch currents at a corresponding solution. By Tellegen's theorem [8], or conservation of complex power, we have

$$\sum_{\text{resistances}} V_k^2/R_k + \sum_{\text{capacitances}} s_0 C_k V_k^2 + \sum_{\substack{\text{cell} \\ \text{output branches}}} Y_n(s_0) V_n^2 = 0. \quad (2)$$

Solutions of the form $s_0 = j\omega \neq 0$ can be ruled out as follows. Note that for each $\omega>0$ all the cell admittance values $Y_n(j\omega)$ lie strictly above and to the right of a straight line through the origin of the complex plane making an angle of $\theta(\omega) - 90°$ with the real positive axis. The capacitance admittances $\{j\omega C_k\}$ and the resistor admittances $\{R_k^{-1}\}$ also lie above and to the right of this line. Thus no positive linear combination of these admittances can vanish as required by eq. (2).

To rule out solutions in the open right half plane, it is shown by a homotopy argument that the existence of such a solution implies the existence of a network satisfying the conditions of Thm. 1 and having natural frequencies of the form $s_0 = j\omega \neq 0$ (already shown not to exist). Add a parallel conductance G to each element of the network, and call the parallel element pair a "composite element." Consider the locus of the natural frequencies as G is increased from zero to arbitrarily high values. Eventually they must all enter the open left half plane because all the composite elements become strictly passive at sufficiently high G values. Since the network started out with at least one open right half plane natural frequency, and the natural frequencies depend continuously on G, then there exists a G>0 such that the network has natural frequencies of the form $s_0 = j\omega \neq 0$ ($s_0=0$ is ruled out by the strict passivity of all the composite elements here). It is easily verified that the collection of composite network elements satisfies the Thm. 1 conditions. Thus, open right half plane natural frequencies are ruled out. ∎

## IV. STABILITY RESULT FOR NETWORKS WITH NONLINEAR RESISTORS AND CAPACITORS

The previous result for linear networks can afford some limited insight into the behavior of nonlinear networks. First the nonlinear equations are linearized about an equilibrium point and Theorem 1 is applied to the linear model. If the linearized model is stable, then the equilibrium point of the original nonlinear network is *locally stable*, i.e., the network will return to that equilibrium point if the initial condition is sufficiently near it. But the result in this section, in contrast, applies to the full nonlinear circuit model and allows one to conclude that in certain circumstances the network cannot oscillate *even if* the initial state is arbitrarily far from the equilibrium point.

Terminology

We say that a function y=f(x) lies in the *sector* $[a,b]$ if $a\cdot x^2 \leq x\cdot f(x) \leq b\cdot x^2$. And we say that an impedance Z(s) satisfies the *Popov criterion* if $(1 + rs)Z(s)$ is *positive real* [7,9,10] for some r>0. Note that this statement of the Popov criterion differs slightly from that given in standard references [9,10].

Theorem 2

Consider a network consisting of possibly nonlinear resistors and capacitors and cells with linear output impedances $Z_n(s)$, n=1,2,...,N. Suppose

i) the resistor curves are continuous functions $i_k = g_k(v_k)$ where $g_k$ lies in the sector $[0,G_{max}]$, $G_{max}>0$, for all resistors,

ii) the capacitors are characterized by $i_k = C_k(v_k)\dot{v}_k$ where $0 \leq C_k(v_k) \leq C_{max}$ for all k and $v_k$, and

iii) the impedances $Z_n(s)$ all satisfy the Popov criterion for some common value of r>0. Then the network is stable in the sense that, for any initial condition,

$$\int_0^\infty \left[ \sum_{\substack{\text{all resistors} \\ \text{and capacitors}}} i_k^2(t) \right] dt < \infty . \quad (3)$$

Outline of Proof

By Tellegen's theorem, for any set of initial conditions and any time T>0,

$$\int_0^T \sum_{\text{resistors}} (v_k(t) + r\,\dot{v}_k(t)) i_k(t)\, dt +$$

$$\int_0^T \sum_{\text{capacitors}} (v_k(t) + r\,\dot{v}_k(t)) i_k(t)\, dt +$$

$$\int_0^T \sum_{\substack{\text{cell} \\ \text{impedances}}} (v_k(t) + r\,\dot{v}_k(t)) i_k(t)\, dt = 0. \quad (4)$$

For resistors, multiplying the sector inequality $v\,g(v) \leq G_{max}v^2$ by $\frac{1}{v} \geq 0$ yields $i^2 = ig(v) \leq G_{max} iv$, and hence

$$G_{max}^{-1} \int_0^T i_k^2(t)\, dt \leq \int_0^T i_k(t)v_k(t)\, dt =$$

$$\int_0^T i_k(t)[v_k(t) + r\,\dot{v}_k(t)]dt - r[\mathfrak{I}_k(v_k(t)) - \mathfrak{I}_k(v_k(0))] \quad (5)$$

where

$$\mathcal{P}_k(v) = \int_0^v i_k(v')dv' \geq 0 \qquad (6)$$

is the resistor co-content. Using the inequality (6) in (5) yields

$$G_{max}^{-1}\int_0^T i_k^2(t)dt - r\mathcal{P}_k(v_k(0)) \leq \int_0^T i_k(t)[v_k(t)+r\dot{v}_k(t)]dt, \qquad (7)$$

For capacitors, integrating the inequality $i_k^2 = C_k^2(v_k)\dot{v}_k^2 \leq C_{max}C_k(v_k)\dot{v}_k^2$ yields

$$\frac{r}{C_{max}}\int_0^T i_k^2(t)dt \leq r\int_0^T C_k(v_k)\dot{v}_k^2(t)dt =$$

$$\int_0^T i_k(t)\,[v_k(t) + r\dot{v}_k(t)]dt - [E_k(q_k(T)-E_k(q_k(0))], \qquad (8)$$

where

$$E_k(q) = \int_0^q v_k(q')dq' \geq 0 \qquad (9)$$

is the capacitor energy. Using the inequality (9) in (8) yields

$$\frac{r}{C_{max}}\int_0^T i_k^2(t)dt - E_k(q_k(0) \leq \int_0^T i_k(t)[v_k(t)+r\dot{v}_k(t)]dt. \qquad (10)$$

And for the cells, the assumption that $(1+rs)Z_n(s)$ is positive real implies that

$$\int_0^T i_n(t)\,[v_n(t) + r\dot{v}_n(t)]dt \geq -E_n(0), \qquad (11)$$

where $E_n(0)$ is the "initial energy in the cell's output impedance" at t=0, a function of the initial conditions only. Substituting (7), (10) and (11) into (4) yields

$$G_{max}^{-1}\int_0^T \sum_{\text{resistors}} i_k^2(t)dt + \frac{r}{C_{max}}\int_0^T \sum_{\text{capacitors}} i_k^2(t)dt \leq$$

$$r\sum_{\text{resistors}}\mathcal{P}_k(v_k(0)) + \sum_{\text{capacitors}} E_k(q_k(0)) + \sum_{\text{cells}} E_n(0), \qquad (12)$$

where the right hand side is a function only of the initial conditions. Thus (3) holds. ∎

## V. CONCLUDING REMARKS

The design criteria presented here are simple and practical, though at present their validity is restricted to linear models of the cells. There are several areas of further work to be pursued, one of which is an analysis of the differentiator cell that includes amplifier clipping effects. Others include the synthesis of a compensator for the differentiator cell, an extension of the non-linear result to include impedance multipliers other than the Popov operator, and a waveform bounding analysis of the network which would guarantee adequate convergence after an allotted settling time.


## ACKNOWLEDGEMENT

We sincerely thank Professor Carver Mead of Cal Tech for enthusiastically supporting this work and for making it possible for us to present an early report on it in this conference proceedings. This work was supported by Defense Advanced Research Projects Agency (DoD), through the Office of Naval Research under ARPA Order No. 3872, Contract No. N00014-80-C-0622 and Defense Advanced Research Projects Agency (DARPA) Contract No. N00014-87-K-0825.


## REFERENCES


1. F.S. Werblin, "The Control of Sensitivity in the Retina," Scientific American, Vol. 228, no. 1, Jan. 1983, pp. 70-79.
2. T. Kohonen, Self-Organization and Associative Memory, (vol. 8 in the Springer Series in Information Sciences), Springer Verlag, New York, 1984.
3. M.A. Sivilotti, M.A. Mahowald, and C.A. Mead, "Real Time Visual Computations Using Analog CMOS Processing Arrays," Advanced Research in VLSI - Proceedings of the 1987 Standard Conference, P. Losleben, ed., MIT Press, 1987, pp. 295-312.
4. C.A. Mead, Analog VLSI and Neural Systems, Addison-Wesley, to appear in 1988.
5. J. Hutchinson, C. Koch, J. Luo and C. Mead, "Computing Motion Using Analog and Binary Resistive Networks," submitted to IEEE Transactions on Computers, August 1987.
6. M. Mahowald, personal communication.
7. B.D.O. Anderson, and S. Vongpanitlerd, Network Analysis and Synthesis - A Modern Systems Theory Approach, Prentice-Hall, Englewood Cliffs, NJ., 1973.
8. P. Penfield, Jr., R. Spence, and S. Duinker, Tellegen's Theorem and Electrical Networks, MIT Press, Cambridge, MA 1970.
9. M. Vidyasagar, Nonlinear Systems Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1970, pp. 211-217.
10. C. Desoer and M. Vidyasagar, Feedback Systems: Input-Output Properties, Academic Press, New York, 1975.



VLSI Memo No. 88-440
March 1988

# REDUCING THE PARALLEL SOLUTION TIME OF SPARSE CIRCUIT MATRICES USING REORDERED GAUSSIAN ELIMINATION AND RELAXATION

David Smart and Jacob White

Abstract

Using parallel processors to reduce the execution times of classical circuit simulation programs like SPICE and ASTAP has been the focus of much current research. In these efforts, good parallel speed increases have been achieved for linearized system construction, but it has been difficult to get good parallel speed increases for sparse matrix solution. In this paper we examine two approaches for reducing parallel sparse matrix solution time; the first based on pivot ordering algorithms for Gaussian elimination, and the second based on relaxation algorithms. In the section on Gaussian elimination sparse matrix solution, we present a pivot ordering algorithm which increases the parallelism of Gaussian elimination compared to the commonly used Markowitz method. The performance of the new algorithm is compared to other suggested ordering algorithms for a collection of circuit examples. The minimum number of parallel steps for the solution of a tridiagonal matrix is derived, and it is shown that this optimum is nearly achieved by the ordering heuristics which attempt to maximize parallelism. In the section on relaxation, we present an optimality result about Gauss-Jacobi over Gauss-Seidel relaxation on parallel processors.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138



VLSI Memo No. 88-441
March 1988

# WAVEFORM RELAXATION APPLIED TO TRANSIENT DEVICE SIMULATION

M. Reichelt, J. White, J. Allen, and F. Odeh

Abstract

In this paper we investigate the possibility of accelerating the transient simulation of MOS devices by using waveform relaxation. Standard spatial discretization techniques are used to generate a large, sparsely-connected system of algebraic and ordinary differential equations in time. The waveform relaxation (WR) algorithm for solving such a system is described, and several theoretical results that characterize the convergence of WR for device simulation are given. In addition, one-dimensional experimental results are presented.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138



VLSI Memo No. 87-427
November 1987

# OPTIMAL SIMULATIONS BY BUTTERFLY NETWORKS: EXTENDED ABSTRACT

Sandeep N. Bhatt, Fan R. K. Chung, Jia-Wei Hong, F. Thomas Leighton and Arnold Rosenberg

Abstract

We investigate the power of the Butterfly network (which is the FFT network with inputs and outputs identified) relative to other proposed multicomputer interconnection networks, by considering how efficiently the Butterfly can simulate the other networks: Formally we ask, How efficiently can one embed the graph underlying the other network in the graph underlying the Butterfly? We measure the efficiency of an embedding of a graph $G$ in a graph $H$ in terms of: the *dilation*, or, the maximum amount that any edge of $G$ is "stretched" by the embedding; the *expansion*, or, the ratio of the number of vertices of $H$ to the number of vertices of $G$. We present three simulations that are optimal, to within constant factors: (1) Any complete binary tree can be embedded in a Butterfly graph, with simultaneous dilation $O(1)$ and expansion $O(1)$. (2) The $n$-vertex X-tree can be embedded in a Butterfly graph with simultaneous dilation $O(\log \log n)$ and expansion $O(1)$; no embedding has better dilation, independent of expansion. (3) Any embedding of the $n \times n$ mesh in the Butterfly graph must have dilation $(\log n)$, independent of expansion; any embedding of the mesh in the Butterfly graph achieves this dilation. Thus, we have simulations of complete-binary-tree machines, X-tree machines, and mesh computers on Butterfly machines, that are optimal in resource utilization (expansion) and delay (dilation), to within constant factors.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138



VLSI Memo No. 87-424
November 1987

# PERFORMANCE ANALYSIS OF *K*-ARY *N*-CUBE INTERCONNECTION NETWORKS

William J. Dally

## Abstract

VLSI communication networks are wire limited. The cost of a network is not a function of the number of switches required, but rather a function of the wiring density required to construct the network. This paper analyzes communication networks of varying dimension under the assumption of constant wire bisection. Expressions for the latency, average case throughput, and hot-spot throughput of *k*-ary *n*-cube networks with constant bisection are derived that agree closely with experimental measurements. It is shown that low-dimensional networks (e.g., tori) have lower latency and higher hot-spot throughput than high-dimensional networks (e.g., binary *n*-cubes) with the same bisection width.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone
(617) 253-8138

# Performance Analysis of $k$-ary $n$-cube Interconnection Networks[1][2]

William J. Dally
Artificial Intelligence Laboratory and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

*Submitted to IEEE Transactions on Computers*

## Abstract

VLSI communication networks are wire limited. The cost of a network is not a function of the number of switches required, but rather a function of the wiring density required to construct the network. This paper analyzes communication networks of varying dimension under the assumption of constant wire bisection. Expressions for the latency, average case throughput, and hot-spot throughput of $k$-ary $n$-cube networks with constant bisection are derived that agree closely with experimental measurements. It is shown that low-dimensional networks (*e.g.*, *tori*) have lower latency and higher hot-spot throughput than high-dimensional networks (*e.g.*, binary $n$-cubes) with the same bisection width.

## Keywords

Communication networks, interconnection networks, concurrent computing, message-passing multiprocessors, parallel processing, VLSI.

## 1 Introduction

The critical component of a concurrent computer is its communication network. Many algorithms are communication rather than processing limited. Fine-grain concurrent programs execute as few as 10 instructions in response to a message [5]. To efficiently execute such programs the communication network must have a latency no greater than about 10 instruction times, and a throughput sufficient to permit a large fraction of the nodes to transmit simultaneously. Low-latency communication is also critical to support code sharing and garbage collection across nodes.

[1]The research described in this paper was supported in part by the Defense Advanced Research Projects Agency under contracts N00014-80-C-0622 and N00014-85-K-0124 and in part by a National Science Foundation Presidential Young Investigator Award with matching funds from General Electric Corporation.

[2]A preliminary version of this paper appeared in the proceedings of the 1987 Stanford Conference on Advanced Research in VLSI [6].

As the grain size of concurrent computers continues to decrease, communication latency becomes a more important factor. The diameter of the machine grows, messages are sent more frequently, and fewer instructions are executed in response to each message. Low latency is more difficult to achieve in a fine-grain machine because the available wiring space grows more slowly than the expected traffic. Since the machine must be constructed in three dimensions, the bisection area grows only as $N^{\frac{2}{3}}$ while traffic grows at least as fast as $N$, the number of nodes.

VLSI systems are wire limited. The cost of these systems is predominantly that of connecting devices, and the performance is limited by the delay of these interconnections. Thus, to achieve the required performance, the network must make efficient use of the available wire. The topology of the network must map into the three physical dimensions so that messages are not required to *double back* on themselves, and in a way that allows messages to use all of the available bandwidth along their path.

This paper considers the problem of constructing *wire-efficient* communication networks, networks that give the optimum performance for a given wire density. We compare networks holding wire bisection, the number of wires crossing a cut that evenly divides the machine, constant. Thus we compare low dimensional networks with wide communication channels against high dimensional networks with narrow channels. We investigate the class of $k$-ary $n$-cube interconnection networks and show that low-dimensional networks out perform high-dimensional networks with the same bisection width.

The remainder of this paper describes the design of wire-efficient communication networks. Section 2 describes the assumptions on which this paper is based. The family of $k$-ary $n$-cube networks is described in Section 2.1. We restrict our attention to $k$-ary $n$-cubes because it is the dimension of the network that is important, not the details of its topology. Section 2.2 introduces *wormhole routing* [18], a low-latency routing technique. Network cost is determined primarily by wire density which we will measure in terms of bisection width. Section 2.3 introduces the idea of *bisection width*, and discusses delay models for network channels. A performance model of these networks is derived in Section 3. Expressions are given for network latency as a function of traffic that agree closely with experimental results. Under the assumption of constant wire density, it is shown that low-dimensional networks achieve lower latency and better hot-spot throughput than do high-dimensional networks.

# 2 Preliminaries

## 2.1 $k$-ary $n$-cubes

Many different network topologies have been proposed for use in concurrent computers: trees [4] [13] [19], Benes networks[3], Batcher sorting networks [1], shuffle exchange networks [21], *Omega* networks [12], *indirect* binary $n$-cube or *flip* networks [2] [20], and direct binary $n$-cubes [17], [15], [22]. The binary $n$-cube is a special case of the family of $k$-ary $n$-cubes, cubes with $n$ dimensions and $k$ nodes in each dimension.

Most concurrent computers have been built using networks that are either $k$-ary $n$-cubes or

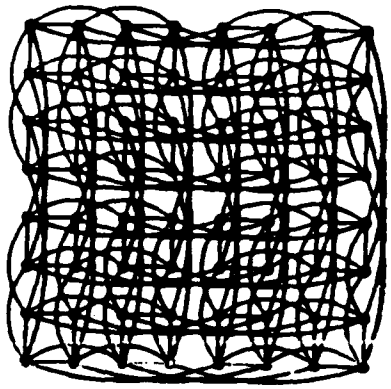


Figure 1: A Binary 6-Cube Embedded in the Plane

are isomorphic to $k$-ary $n$-cubes: rings, meshes, tori, direct and indirect binary $n$-cubes, and Omega networks. Thus, in this paper we restrict our attention to $k$-ary $n$-cube networks. We refer to $n$ as the *dimension* of the cube and $k$ as the *radix*. Dimension, radix, and number of nodes are related by the equation

$$N = k^n, \quad \left(k = \sqrt[n]{N}, \quad n = \log_k N\right). \tag{1}$$

It is the dimension of the network that is important, not the details of its topology.

A node in a $k$-ary $n$-cube can be identified by an $n$-digit radix $k$ address, $a_0, \ldots, a_{n-1}$. The $i^{\text{th}}$ digit of the address, $a_i$, represents the nodes position in the $i^{\text{th}}$ dimension. Each node can forward messages to its upper neighbor in each dimension, $i$, with address, $a_0, \ldots, a_i + 1 (\text{mod } k), \ldots, a_{n-1}$.

In this paper we assume that our $k$-ary $n$-cube are unidirectional for simplicity. We will see that our results do not change appreciably for bidirectional networks. For an actual machine, however, there are many compelling reasons to make our networks bidirectional. Most importantly, bidirectional networks allow us to exploit locality of communication. If an object, $A$, sends a message to an object, $B$, there is a high probability of $B$ sending a message back to $A$. In a bidirectional network, a round trip from $A$ to $B$ can be made short by placing $A$ and $B$ close together. In a unidirectional network, a round trip will always involve completely circling the machine in at least one dimension.

Figures 1-3 show three $k$-ary $n$-cube networks in order of decreasing dimension. Figure 1 shows a binary 6-cube (64 nodes). A 3-ary 4-cube (81 nodes) is shown in Figure 2. An 8-ary 2-cube (64 nodes), or torus, is shown in Figure 3. Each line in Figure 1 represents two communication channels, one in each direction, while each line in Figures 2 and 3 represents a single communication channel.

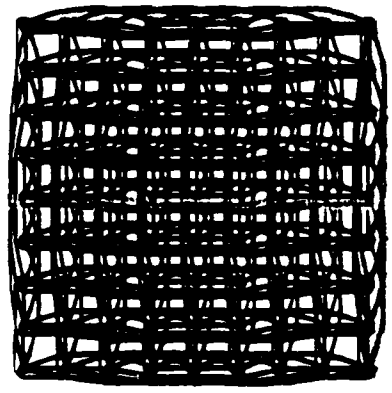


Figure 2: A Ternary 4-Cube Embedded in the Plane

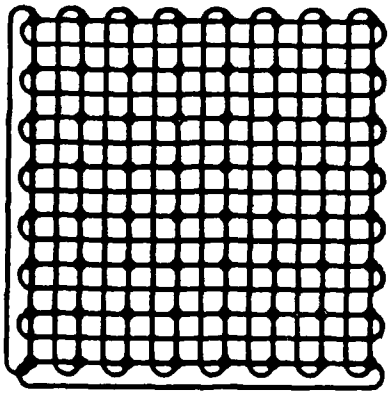


Figure 3: An 8-ary 2-Cube (Torus)

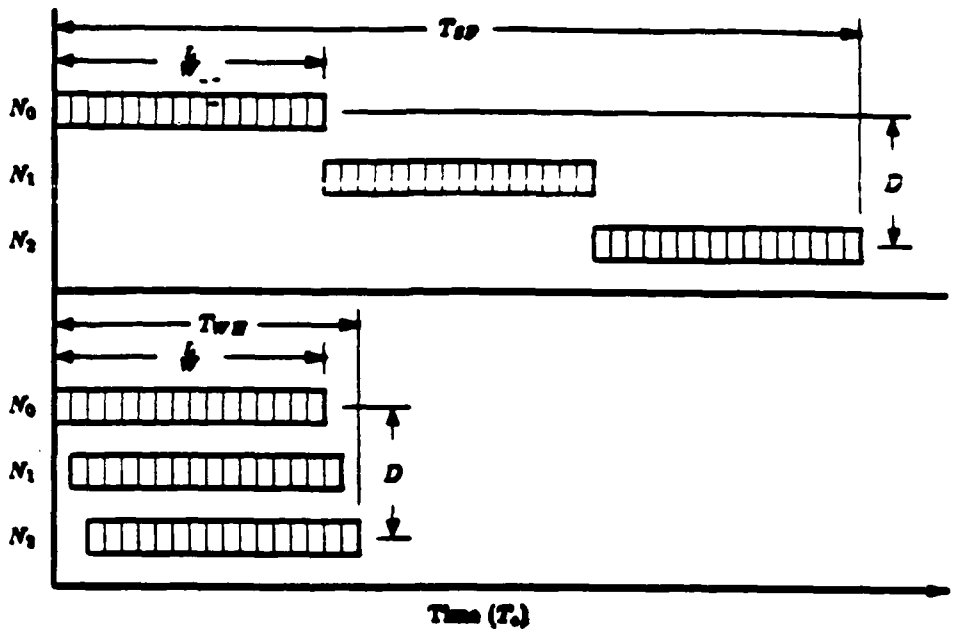



Figure 4: Latency of store-and-forward routing (top) vs. wormhole routing (bottom).

## 2.2 Wormhole Routing

In this paper we consider networks that use *wormhole*[18] rather than *store-and-forward* [23] routing. Instead of storing a packet completely in a node and then transmitting it to the next node, wormhole routing operates by advancing the head of a packet directly from incoming to outgoing channels. Only a few flow control digits (flits) are buffered at each node. A *flit* is the smallest unit of information that a queue or channel can accept or refuse.

As soon as a node examines the header flit(s) of a message, it selects the next channel on the route and begins forwarding flits down that channel. As flits are forwarded, the message becomes spread out across the channels between the source and destination. It is possible for the first flit of a message to arrive at the destination node before the last flit of the message has left the source. Because most flits contain no routing information, the flits in a message must remain in contiguous channels of the network and cannot be interleaved with the flits of other messages. When the header flit of a message is blocked, all of the flits of a message stop advancing and block the progress of any other message requiring the channels they occupy.

A method similar to wormhole routing, called *virtual cut-through*, is described in [11]. Virtual cut-through differs from wormhole routing in that it buffers messages when they block, removing them from the network. With wormhole routing, blocked messages remain in the network.

Figure 4 illustrates the advantage of wormhole routing. There are two components of latency, distance and message aspect ratio. The distance, $D$, is the number of *hops* required to get from the source to the destination. The message aspect ratio (message length, $L$, normalized to the channel width, $W$) is the number of channel cycles required to transmit the message across one channel. The top half of the figure shows store-and-forward routing. The message is is entirely transmitted from node $N_0$ to node $N_1$, then from $N_1$ to $N_2$ and so on. With store-and-forward routing, latency is the product of $D$, and $\frac{L}{W}$.

$$T_{SF} = T_c \left( D \times \frac{L}{W} \right). \tag{2}$$

The bottom half of Figure 4 shows wormhole routing. As soon as a flit arrives at a node, it is forwarded to the next node. With wormhole routing latency is reduced to the sum of $D$ and $\frac{L}{W}$.

$$T_{WH} = T_c \left( D + \frac{L}{W} \right). \tag{3}$$

In both of these equations, $T_c$ is the channel cycle time, the amount of time required to perform a transaction on a channel.

## 2.3 VLSI Complexity

VLSI computing systems [14] are wire-limited; the complexity of what can be constructed is limited by wire density, the speed at which a machine can run is limited by wire delay, and the majority of power consumed by a machine is used to drive wires. Thus, machines must be organized both logically and physically to keep wires short by exploiting locality wherever possible. The VLSI architect must organize a computing system so that its form (physical organization) fits its function (logical organization).

Networks have traditionally been analyzed under the assumption of constant channel bandwidth. Under this assumption each channel is one bit wide ($W = 1$) and has unit delay ($T_c = 1$). The constant bandwidth assumption favors networks with high dimensionality (e.g., binary $n$-cubes) over low-dimensional networks (e.g., tori). This assumption, however, is not consistent with the properties of VLSI technology. Networks with many dimensions require more and longer wires than do low-dimensional networks. Thus, high-dimensional networks cost more and run more slowly than low-dimensional networks. A realistic comparison of network topology must take both wire density and wire length into account.

To account for wire density, we will use bisection width [24] as a measure of network cost. The bisection width of a network is the minimum number of wires cut when the network is divided into two equal halves. Rather than comparing networks with constant channel width, $W$, we will compare networks with constant bisection width. Thus, we will compare low-dimensional networks with large $W$ with high-dimensional networks with small $W$.

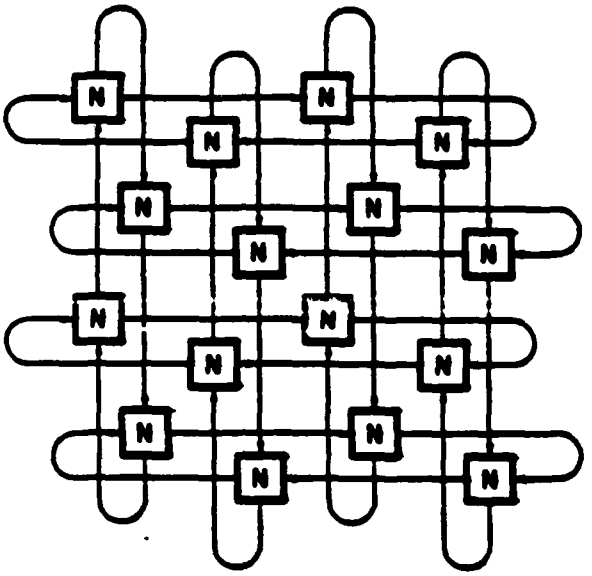



Figure 5: A Folded Torus System

The delay of a wire depends on its length, $l$. For short wires, the delay, $t_s$, is limited by charging the capacitance of the wire and varies logarithmically with wire length.

$$t_s = \tau_{\text{inv}} e \log_e Kl, \tag{4}$$

where $\tau_{\text{inv}}$ is the inverter delay, and $K$ is a constant depending on capacitance ratios.

For long wires, delay, $t_l$, is limited by the speed of light.

$$t_l = \frac{l\sqrt{\epsilon_r}}{c} \tag{5}$$

In this paper we will consider three delay models: constant delay, $T_c$ independent of length, logarithmic delay, $T_c \propto \log l$, and linear delay, $T_c \propto l$. Our main result, that latency is minimized by low-dimensional networks, is supported by all three models.

# 3 Performance Analysis

In this section we compare the performance of unidirectional $k$-ary $n$-cube interconnection networks using the following assumptions:

- Networks must be embedded into the plane. If a three-dimensional packaging technology becomes available, the comparison changes only slightly.

- Nodes are placed systematically by embedding $\frac{n}{2}$ logical dimensions in each of the two physical dimensions. We assume that both $n$ and $k$ are even integers. The long end-around connections shown in Figure 3 can be avoided by folding the network as shown in Figure 5.
- For networks with the same number of nodes, *wire density is held constant.* Each network is constructed with the same bisection width, $B$, the total number of wires crossing the midpoint of the network. To keep the bisection width constant, we vary the width, $W$, of the communication channels. We normalize to the bisection width of a bit-serial ($W = 1$) binary $n$-cube.
- The networks use *wormhole* routing.
- Channel delay, $T_c$, is a function of wire length, $l$. We begin by considering channel delay to be constant. Later, the comparison is performed for both logarithmic and linear wire delays; $T_c \propto \log l$ and $T_c \propto l$.

When $k$ is even, the channels crossing the midpoint of the network are all in the highest dimension. For each of the $\sqrt{N}$ rows of the network, there are $k^{(\frac{n}{2}-1)}$ of these channels in each direction for a total of $2\sqrt{N}k^{(\frac{n}{2}-1)}$ channels. Thus, the bisection width, $B$, of a $k$-ary $n$-cube with $W$-bit wide communication channels is

$$B(k,n) = 2W\sqrt{N}k^{(\frac{n}{2}-1)} = \frac{2WN}{k}. \tag{6}$$

For a binary $n$-cube, $k = 2$, the bisection width is $B(2,n) = WN$. We set $B$ equal to $N$ to normalize to a binary $n$-cube with unit width channels, $W = 1$. The channel width, $W(k,n)$, of a $k$-ary $n$-cube with the same bisection width, $B$, follows from (6):

$$\begin{aligned} \tfrac{2W(k,n)N}{k} &= N, \\ W(k,n) &= \tfrac{k}{2}. \end{aligned} \tag{7}$$

The peak wire density is greater than the bisection width in networks with $n > 2$ because the lower dimensions contribute to wire density. The maximum density, however, is bounded by

$$\begin{aligned} D_{\max} &= 2W\sqrt{N}\sum_{i=0}^{\frac{n}{2}-1} k^i = k\sqrt{N}\sum_{i=0}^{\frac{n}{2}-1} k^i = k\sqrt{N}\left(\frac{k^{\frac{n}{2}}-1}{k-1}\right) \\ &= k\sqrt{N}\left(\frac{\sqrt{N}-1}{k-1}\right) < \left(\frac{k}{k-1}\right)B. \end{aligned} \tag{8}$$

A plot of wire density as a function of position for one row of a binary 20-cube is shown in Figure 6. The density is very low at the edges of the cube and quite dense near the center.

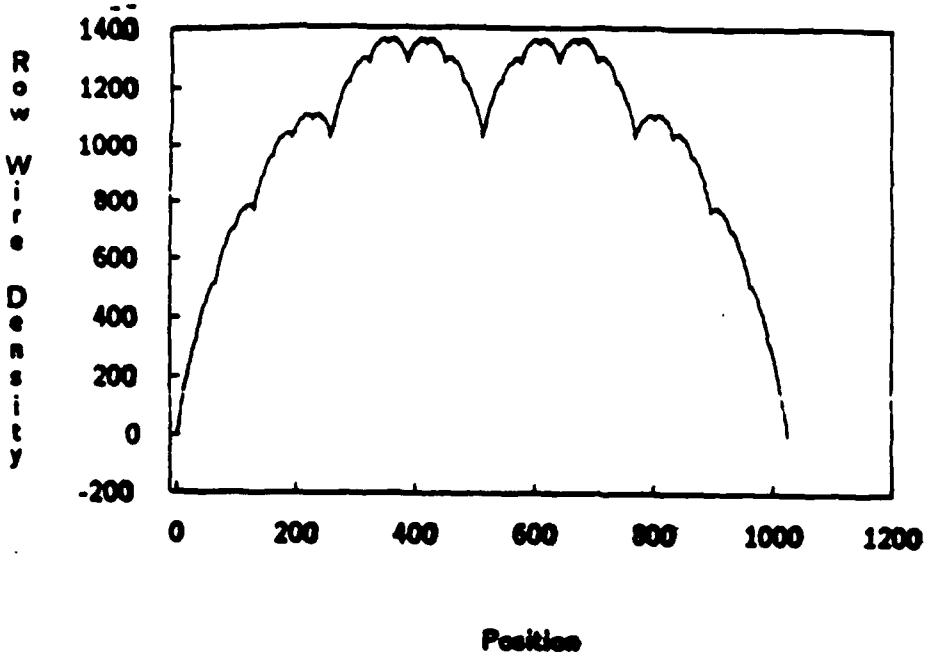



Figure 6: Wire Density vs. Position for One Row of a Binary 20-Cube

The peak density for the row is 1364 at position 341. Compare this density with the bisection width of the row, which is 1024. In contrast, a two-dimensional torus has a wire density of 1024 independent of position. One advantage of high-radix networks is that they have a very uniform wire density. They make full use of available area.

Each processing node connects to $2n$ channels ($n$ input and $n$ output) each of which is $\frac{k}{2}$ bits wide. Thus, the number of pins per processing node is

$$N_p = nk. \tag{9}$$

A plot of pin density as a function of dimension for $N = 256$, 16K and 1M nodes[3] is shown in Figure 7. Low-dimensional networks have the disadvantage of requiring many pins per processing node. A two-dimensional network with $1M$ nodes (not shown) requires 2048 pins and is clearly unrealizable. However, the number of pins decreases very rapidly as the dimension, $n$, increases. Even for 1M nodes, a dimension 4 node has only 128 pins. All of the configurations that give low latency also give a reasonable pin count.

## 3.1 Latency

Latency, $T_l$, is the sum of the latency due to the network and the latency due to the processing node,

$$T_l = T_{\text{net}} + T_{\text{node}}. \tag{10}$$

[3] $1K = 1024$ and, $1M = 1K \times 1K = 1048576$.

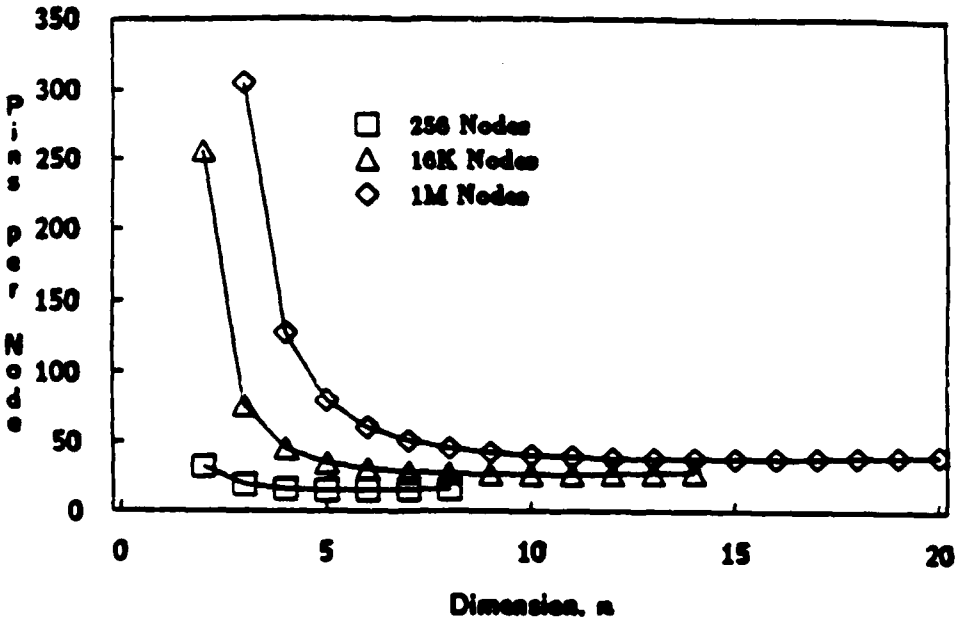



Figure 7: Pin Density vs. Dimension for 256, 16K, and 1M Nodes

In this paper we are concerned only with $T_{net}$. Techniques to reduce $T_{node}$ are described in [5] and [9].

If we select two processing nodes, $P_i, P_j$, at random, the average number of channels that must be traversed to send a message from $P_i$ to $P_j$ is given by

$$D = \left(\frac{k-1}{2}\right) n. \tag{11}$$

The average latency of a $k$-ary $n$-cube is calculated by substituting (7) and (11), into (3)

$$T_{net} = T_c \left( \left(\frac{k-1}{2}\right) n + \frac{2L}{k} \right). \tag{12}$$

Figure 8 shows the average network latency, $T_{net}$, as a function of dimension, $n$, for $k$-ary $n$-cubes with $2^8$ (256), $2^{14}$ (16K), and $2^{20}$ (1M) nodes[4]. The left most data point in this figure corresponds to a torus ($n = 2$) and the right most data point corresponds to a binary $n$-cube ($k = 2$). This figure assumes constant wire delay, $T_c$, and a message length, $L$, of 150 bits. This choice of message length was based on the analysis of a number of fine-grain concurrent programs [5]. Although constant wire delay is unrealistic, this figure illustrates that even ignoring the dependence of wire delay on wire length, low-dimensional networks achieve lower latency than high-dimensional networks.

[4]For the sake of comparison we allow radix to take on non-integer values. For some of the dimensions considered, there is no integer radix, $k$, that gives the correct number of nodes. In fact, this limitation can be overcome by constructing a *mixed-radix cube*.

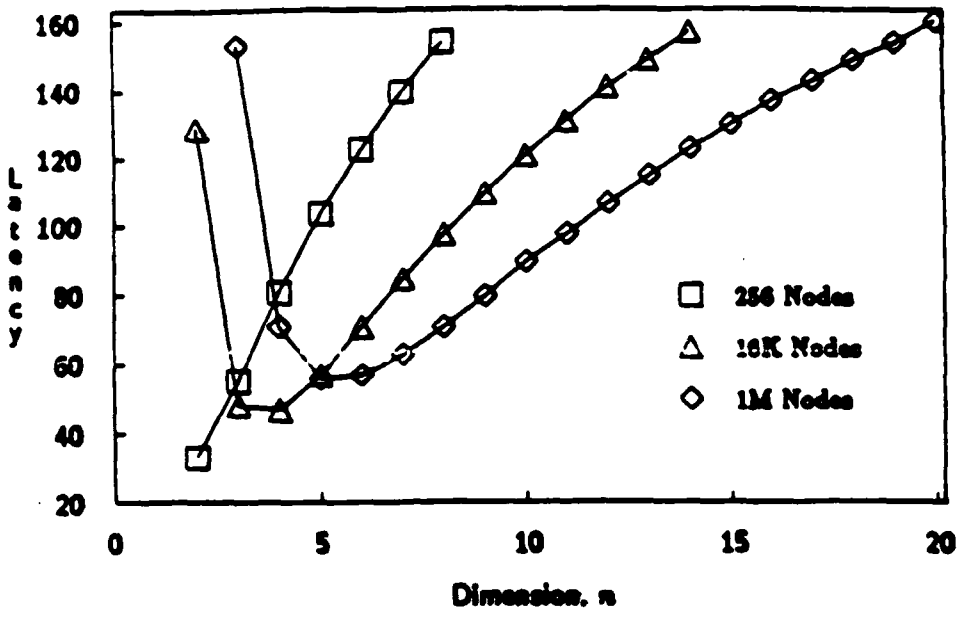



Figure 8: Latency vs. Dimension for 256, 16K, and 1M Nodes, Constant Delay

The latency of the tori on the left side of Figure 8 is limited almost entirely by distance. The latency of the binary $n$-cubes on the right side of the graph is limited almost entirely by aspect ratio. With bit serial channels, these cubes take 150 cycles to transmit their messages across a single channel.

In an application that exploits locality of communication, the distance between communicating objects is reduced. In such a situation, the latency of the low-dimensional networks (the left side of Figure 8) is reduced. High-dimensional networks, on the other hand, cannot take advantage of locality. Their latency will remain high.

In applications that send short messages, the component of latency due to message length is reduced resulting in lower latency for high-dimensional networks (the right side of Figure 8).

In general the lowest latency is achieved when the component of latency due to distance, $D$, and the component due to message length, $\frac{L}{W}$, are approximately equal, $D \approx \frac{L}{W}$. For the three cases shown in Figure 8, minimum latencies are achieved for $n = 2$, 4, and 5 respectively.

The longest wire in the system becomes a bottleneck that determines the rate at which each channel operates, $T_c$. The length of this wire is given by

$$l = k^{\frac{n}{2}-1}. \tag{13}$$

If the wires are sufficiently short, delay depends logarithmically on wire length. If the channels are longer, they become limited by the speed of light, and delay depends linearly on channel

length. Substituting (13) into (4) and (5) gives

$$T_c \propto \begin{cases} 1 + \log_e l = 1 + \left(\frac{n}{2} - 1\right) \log_e k & \text{logarithmic delay} \\ \\ l = k^{\frac{n}{2}-1} & \text{linear delay.} \end{cases} \tag{14}$$

We substitute (14) into (12) to get the network latency for these two cases:

$$T_l \propto \begin{cases} \left(1 + \left(\frac{n}{2} - 1\right) \log_e k\right) \left(\left(\frac{k-1}{2}\right) n + \frac{2L}{k}\right) & \text{logarithmic delay} \\ \\ \left(k^{\frac{n}{2}-1}\right) \left(\left(\frac{k-1}{2}\right) n + \frac{2L}{k}\right) & \text{linear delay.} \end{cases} \tag{15}$$

Figure 9 shows the average network latency as a function of dimension for $k$-ary $n$-cubes with $2^8$ (256), $2^{14}$ (16K), and $2^{20}$ (1M) nodes, assuming logarithmic wire delay and a message length, $L$, of 150. Figure 10 shows the same data assuming linear wire delays. In both figures, the left most data point corresponds to a torus ($n = 2$) and the right most data point corresponds to a binary $n$-cube ($k = 2$).

In the linear delay case, Figure 10, a torus ($n = 2$) always gives the lowest latency. This is because a torus offers the highest bandwidth channels and the most direct physical route between two processing nodes. Under the linear delay assumption, latency is determined solely by bandwidth and by the physical distance traversed. There is no advantage in having long channels.

Under the logarithmic delay assumption, Figure 9, a torus has the lowest latency for small networks ($N = 256$). For the larger networks, the lowest latency is achieved with slightly higher dimensions. With $N = 16K$, the lowest latency occurs when $n$ is three[5]. With $N = 1M$, the lowest latency is achieved when $n$ is 5. It is interesting that assuming constant wire delay does not change this result much. Recall that under the (unrealistic) constant wire delay assumption, Figure 8, the minimum latencies are achieved with dimensions of 2, 4, and 5 respectively.

The results shown in Figures 9 through 8 were derived by comparing networks under the assumption of constant wire cost to a binary $n$-cube with $W = 1$. For small networks it is possible to construct binary $n$-cubes with wider channels, and for large networks (e.g., $1M$ nodes) it may not be possible to construct a binary $n$-cube at all. The available wiring area grows as $N^{\frac{2}{3}}$ while the bisection width of a binary $n$-cube grows as $N$. In the case of small networks, the comparison against binary $n$-cubes with wide channels can be performed by expressing message length in terms of the binary $n$-cube's channel width, in effect decreasing the message length for purposes of comparison. The net result is the same: lower-dimensional networks give lower latency. Even if we perform the 256 node comparison against a binary $n$-cube with $W = 16$, the torus gives the lowest latency under the logarithmic delay model, and a dimension 3 network gives minimum latency under the constant delay model. For large

[5]In an actual machine the dimension $n$ would be restricted to be an even integer.

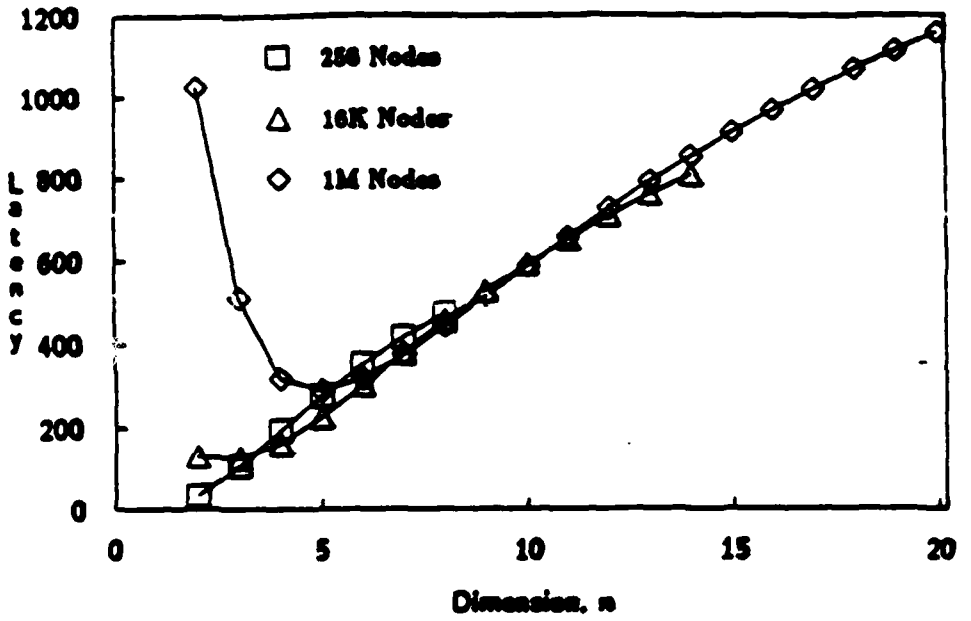



Figure 9: Latency vs. Dimension for 256, 16K, and 1M Nodes, Logarithmic Delay

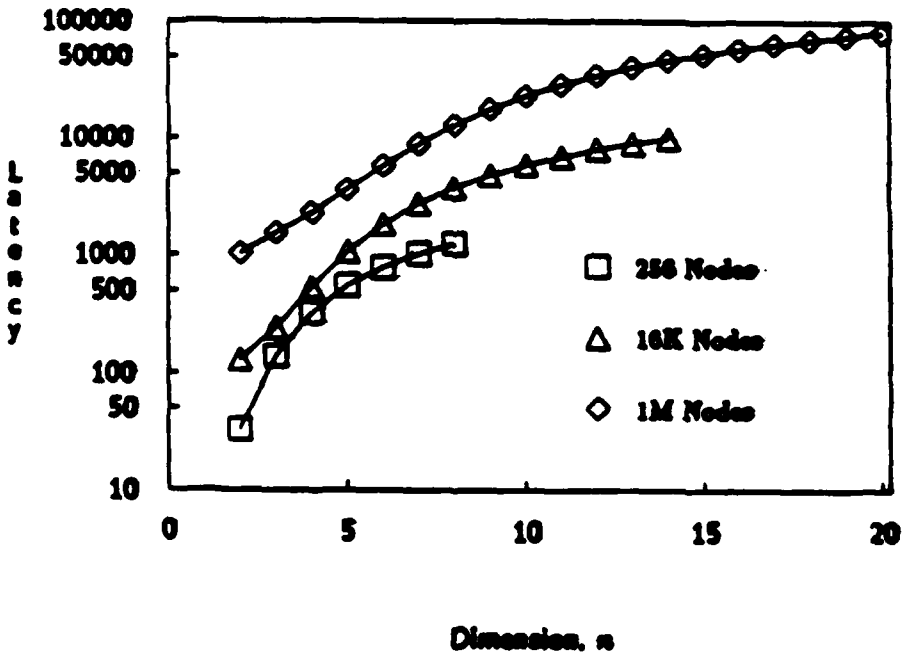



Figure 10: Latency vs. Dimension for 256, 16K, and 1M Nodes, Linear Delay

networks, the available wire is less than assumed, so the effective message length should be increased. Making low-dimensional networks look even more favorable.

In this comparison we have assumed that only a single bit of information is in transit on each wire of the network at a given time. Under this assumption, the delay between nodes, $T_c$, is equal to the period of each node, $T_p$. In a network with long wires, however, it is possible to have several bits in transit at once. In this case, the channel delay, $T_c$, is a function of wire length, while the channel period, $T_p < T_c$, remains constant. Similarly, in a network with very short wires we may allow a bit to ripple through several channels before sending the next bit. In this case, $T_p > T_c$. Separating the coefficients, $T_c$ and $T_p$, (3) becomes

$$T_{\text{net}} = \left( T_c D + T_p \frac{L}{W} \right). \tag{16}$$

The net effect of allowing $T_c \neq T_p$ is the same as changing the length, $L$, by a factor of $\frac{T_p}{T_c}$ and does not change our results significantly.

When wire cost is considered, low-dimensional networks (e.g., tori) offer lower latency than high-dimensional networks (e.g., binary $n$-cubes). Intuitively, tori outperform binary $n$-cubes because they better match form to function. The logical and physical graphs of the torus are identical; Thus, messages always travel the minimum distance from source to destination. In a binary $n$-cube, on the other hand, the fit between form and function is not as good. A message in a binary $n$-cube embedded into the plane may have to traverse considerably more than the minimum distance between its source and destination.

## 3.2 Throughput

Throughput, another important metric of network performance, is defined as the total number of messages the network can handle per unit time. One method of estimating throughput is to calculate the capacity of a network, the total number of messages that can be in the network at once. Typically the maximum throughput of a network is some fraction of its capacity. The network capacity per node is the total bandwidth out of each node divided by the average number of channels traversed by each message. For $k$-ary $n$-cubes, the bandwidth out of each node is $nW$, and the average number of channels traversed is given by (11), so the network capacity per node is given by

$$\Gamma \propto \frac{nW}{D} \propto \frac{n\left(\frac{k}{2}\right)}{\left(\frac{k-1}{2}\right)n} \approx 1. \tag{17}$$

The network capacity is independent of dimension. For a constant wire density, there is a constant network capacity.

Throughput will be less than capacity because contention causes some channels to block. This contention also increases network latency. To simplify the analysis of this contention, we make the following assumptions:

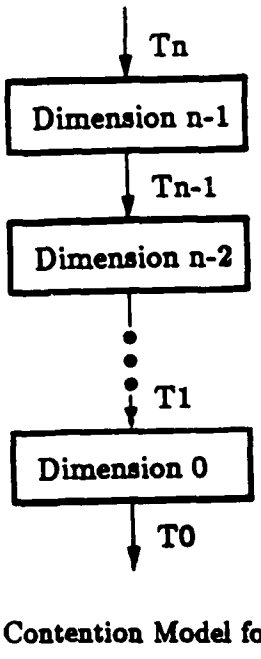



Figure 11: Contention Model for A Network

- Messages are routed using e-cube routing (in order of decreasing dimension) [6]. That is a message at node $a_0, \ldots, a_{n-1}$ destined for node $b_0, \ldots, b_{n-1}$ is first routed in dimension $n-1$ until it reaches node $a_0, \ldots, a_{n-2}, b_{n-1}$. The message is then routed in dimension $n-2$ until it reaches node $a_0, \ldots, a_{n-3}, b_{n-2}, b_{n-1}$, and so on. As shown in Figure 11, this assumption allows us to consider the contention in each dimension separately.
- The traffic from each node is generated by a Poisson process with arrival rate $\lambda \frac{\text{bits}}{\text{cycle}}$.
- Message destinations are uniformly distributed and independent.

The arrival rate of $\lambda \frac{\text{bits}}{\text{cycle}}$ corresponds to $\lambda_E = \frac{\lambda}{L} \frac{\text{messages}}{\text{cycle}}$. At the destination, each flit is serviced as soon as it arrives, so the service time at the sink is $T_0 = \frac{L}{W} = \frac{2L}{k}$. Starting with $T_0$ we will calculate the service time seen entering each preceding dimension.

For convenience, we will define the following quantities:

$$\begin{aligned} \gamma &= \frac{1}{k}, \\ \lambda_S &= \gamma \lambda_E, \\ \lambda_R &= (1-\gamma)\lambda_E, \\ \lambda_{SS} &= \gamma^2 \lambda_E, \\ \lambda_{SR} &= \gamma(1-\gamma)\lambda_E, \\ \lambda_{RS} &= \gamma(1-\gamma)\lambda_E, \text{and} \\ \lambda_{RR} &= (1-\gamma)^2 \lambda_E. \end{aligned} \tag{18}$$

Consider a single dimension, $i$, of the network as shown in Figure 12. All messages incur a latency, $T_E$, due to contention on entering the dimension. Those messages that are routed

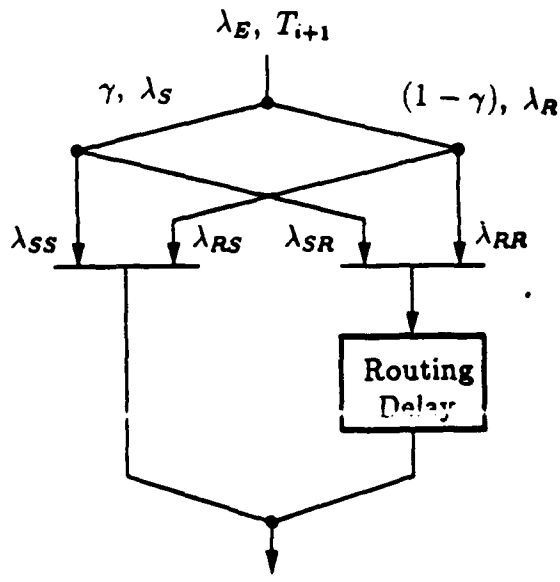



Figure 12: Contention Model for A Single Dimension

incur an additional latency, $T_{Ri}$, due to contention during routing. The rate $\lambda_E$ message stream entering the dimension is composed of two components: a rate $\lambda_S$ stream that skipped the previous ($i+1^{st}$) dimension, and a rate $\lambda_R$ stream that was routed in the previous dimension. These two streams are in turn split into components that will skip the $i^{th}$ dimension ($\lambda_{SS}$ and $\lambda_{RS}$) and components that will be routed in the $i^{th}$ dimension ($\lambda_{SR}$ and $\lambda_{RR}$). The entering latency seen by one component (say $\lambda_{RR}$) is given by multiplying the probability of a collision (in this case $\lambda_{SR}T_{i+1}$) by the expected latency due to a collision, (in this case $\frac{T_i+T_{Ri}}{2}$). The components that require routing must also add the latency due to contention during routing, $T_{Ri}$. Adding up the four components with appropriate weights gives the following equation for $T_{i+1}$.

$$T_{i+1} = T_i + (1-\gamma)T_{Ri} + \gamma(1-\gamma)^3\lambda_E(T_i + T_{Ri}) + \gamma^3(1-\gamma)\lambda_E T_i. \tag{19}$$

For large $k$, *gamma* is small and the latency is approximated by $T_{i+1} \approx T_i + T_{Ri}$. For $k = 2$ (binary $n$-cubes), $T_{Ri} = 0$; thus, $T_{i+1} = T_i + \frac{\lambda_E T_i}{8}$.

To calculate the routing latency, $T_{Ri}$, we use the model shown in Figure 13. Given that a message is to be routed in a dimension, the expected number of channels traversed by the message is $\frac{k}{2}$, one entering channel and $\sigma = \frac{k-2}{2}$ continuing channels. Thus, the average message rate on channels continuing in the dimension is $\lambda_C = \sigma\lambda_R$. Using virtual channels and e-cube routing, the actual continuting rate on the $j^{th}$ channel (outer spiral) is $\lambda_C j = (j - \frac{i^2+i}{2k})\lambda_R$. To calculate $T_R$ we need only the average rate.

The service time in the last continuing channel in dimension $i$ is $T_{i(\sigma-1)} = T_i$. Once we know the service time for the $j^{th}$ channel, $T_{ij}$, the additional service time due to contention at the $j-1^{st}$ channel is given by multiplying the probability of a collision, $\lambda_R T_{i0}$, by the expected waiting time for a collision, $\frac{T_{i0}}{2}$. Repeating this calculation $\sigma$ times gives us $T_{i0}$.

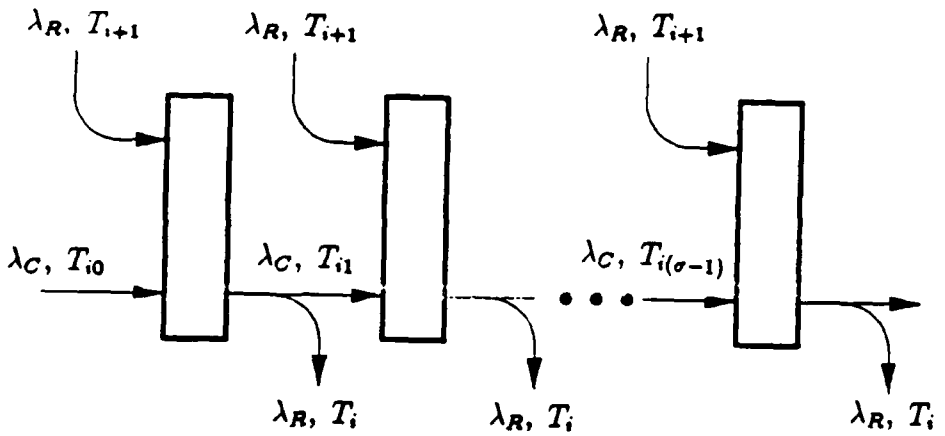



Figure 13: Contention Model for Routing Latency

$$
\begin{aligned}
T_{i(j-1)} &= T_{ij} + \frac{\lambda_R T_{i0}^2}{2}, \\
T_{i0} &= T_i + \frac{\sigma \lambda_R T_{i0}^2}{2} = T_i + \frac{\lambda_C T_{i0}^2}{2}, \\
&= \frac{1 - \sqrt{1 - 2\lambda_C T_i}}{\lambda_C}.
\end{aligned}
\tag{20}
$$

Equation (20) is valid only when $\lambda_C < \frac{1}{2}$. If the message rate is higher than this limit, there is no steady-state solution and latency becomes infinite. There are two solutions to (20). Here we consider only the smaller of the two latencies. The larger solution corresponds to a state that is not encountered during normal operation of a network.

To calculate $T_{Ri}$ we also need to consider the possibility of a collision on the entering channel.

$$
T_{Ri} = T_{i0}\left(1 + \frac{\lambda_C T_{i0}}{2}\right) - T_i. \tag{21}
$$

If sufficient queueing is added to each network node, the service times do not increase, only the latency and equations (21) and (19) become.

$$
T_{Ri} = \left(\frac{T_i}{1 - \frac{\lambda_C T_0}{2}}\right)\left(1 + \frac{\lambda_C T_0}{2}\right) - T_i, \tag{22}
$$

$$
T_{i+1} = T_i + (1-\gamma)T_{Ri} + \left(\gamma(1-\gamma)^3 + \gamma^3(1-\gamma)\right)\lambda_E T_0. \tag{23}
$$

To be effective, the total queueing between the source and destination should be greater than

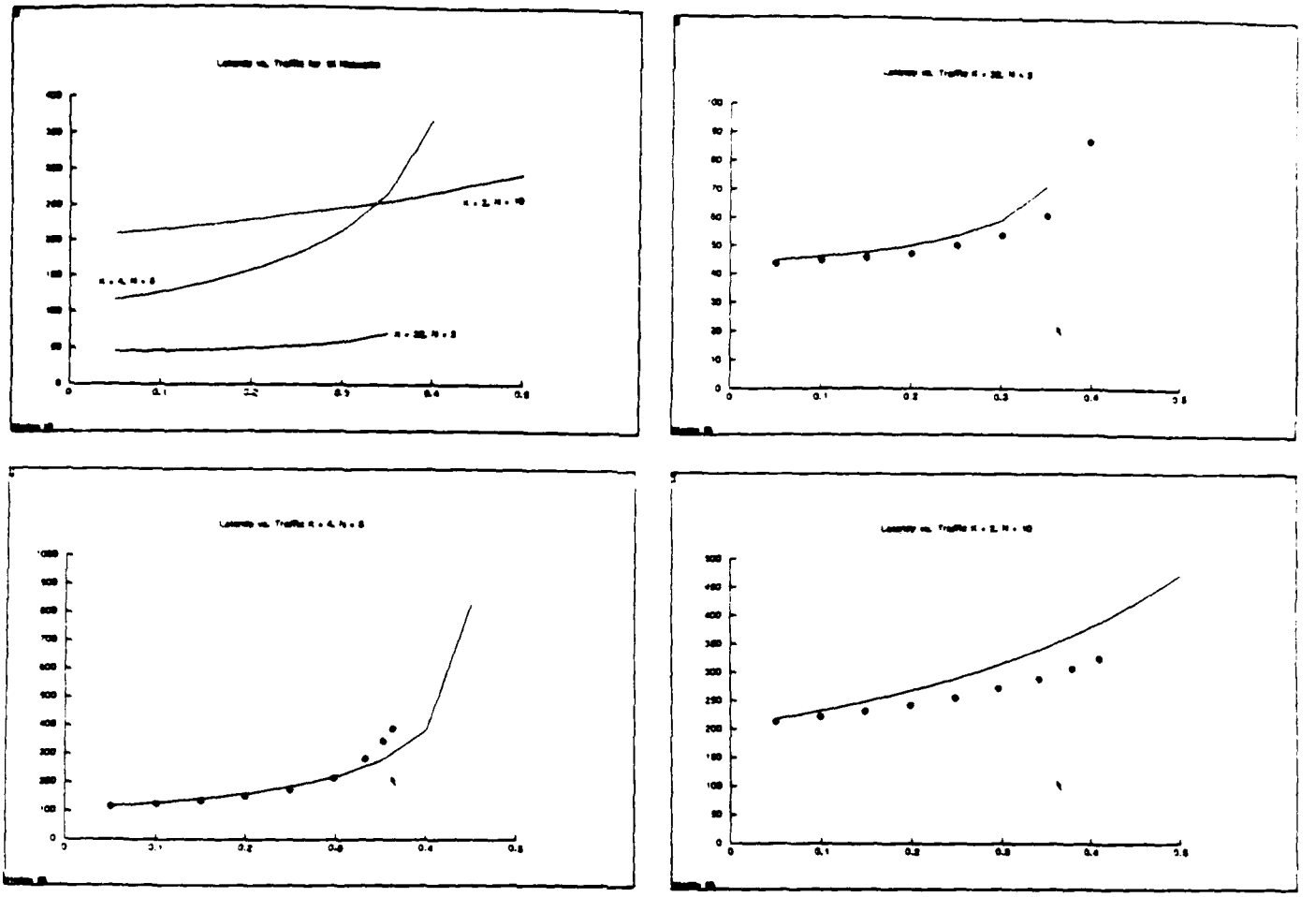


Figure 14: Latency vs. Traffic ($\lambda$) for 1K node networks: 32-ary 2-cube, 4-ary 5-cube, and binary 10-cube, L=200bits. Solid line is predicted latency, points are measurements taken from a simulator.

the expected increase in latency due to blocking. One or two flits of queueing per stage is usually sufficient. The analysis here is pessimistic in that it assumes no queueing.

Using equation (19), we can determine (1) the maximum throughput of the network and (2) how network latency increases with traffic.

Figures 14 and 15 show how latency increases as a function of applied traffic for 1K node and 4K node $k$-ary $n$-cubes. The vertical axis shows latency in cycles. The horizontal axis is traffic per node, $\lambda$, in bits/cycle. The figures compare measurements from a network simulator (points) to the latency predicted by (21) (lines). The simulation agrees with the prediction within a few percent until the network approaches saturation.

For 1K networks, a 32-ary 2-cube always gives the lowest latency. For 4K networks, a 16-ary 3-cube gives the lowest latency when $\lambda < 0.2$. Because latency increases more slowly for 2-dimensional networks, a 64-ary 2-cube gives the lowest latency when $\lambda > 0.2$.

At the left side of each graph ($\lambda = 0$), latency is given by (12). As traffic is applied to the network latency increases slowly due to contention in the network until saturation is reached. Saturation occurs when $\lambda$ is between 0.3 and 0.5 depending on the network topology. Networks should be designed to operate on the flat portion of the curve ($\lambda < 0.25$).

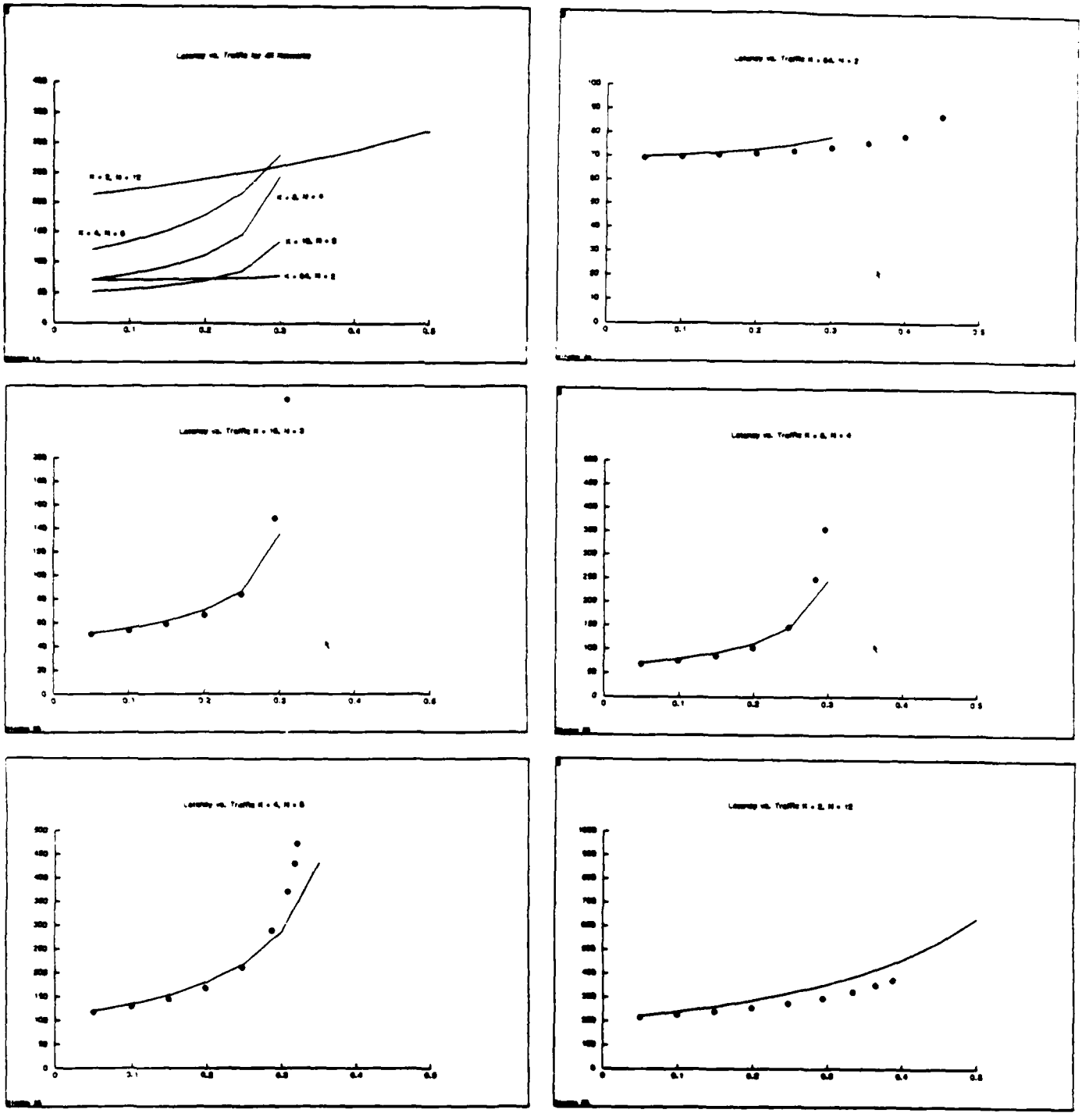


Figure 15: Latency vs. Traffic ($\lambda$) for 4K node networks: 64-ary 2-cube, 16-ary 3-cube, 8-ary 4-cube, 4-ary 6-cube, and binary 12-cube, L=200bits. Solid line is predicted latency, points are measurements taken from a simulator.

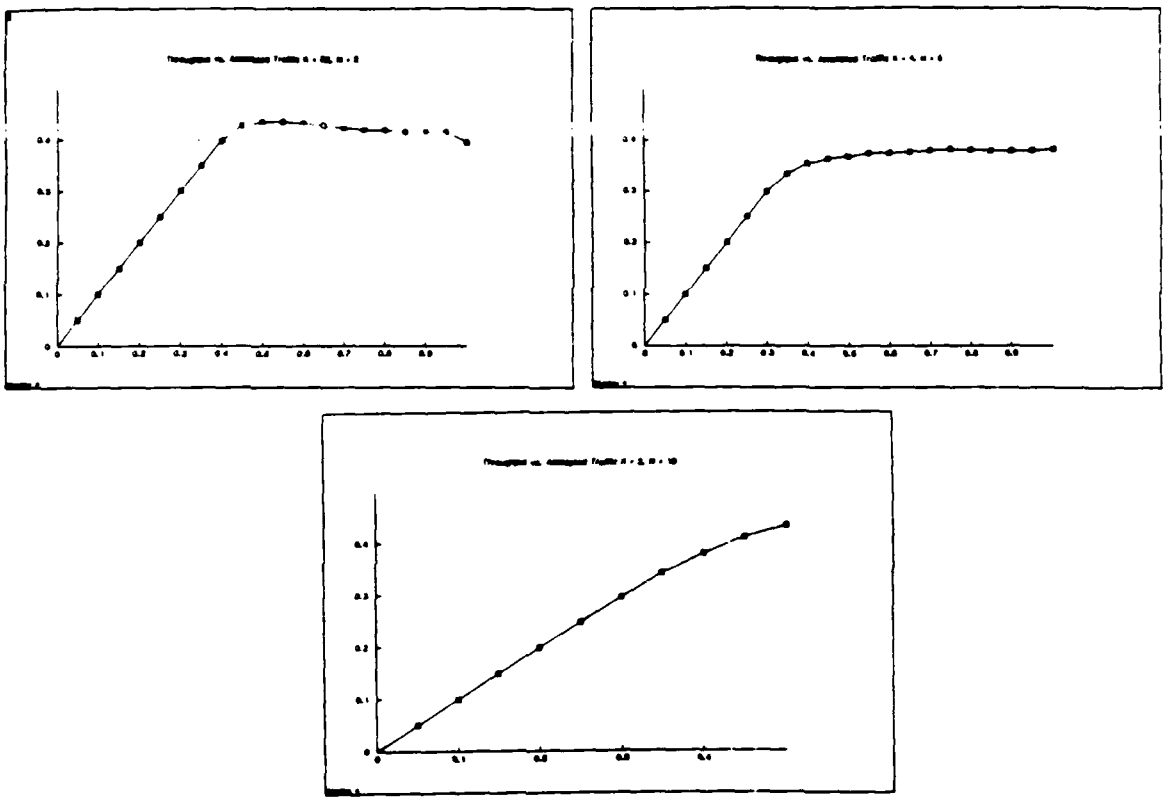


Figure 16: Actual Traffic vs. Attempted Traffic for 1K node networks: 32-ary 2-cube, 4-ary 5-cube, and binary 10-cube, L=200bits.

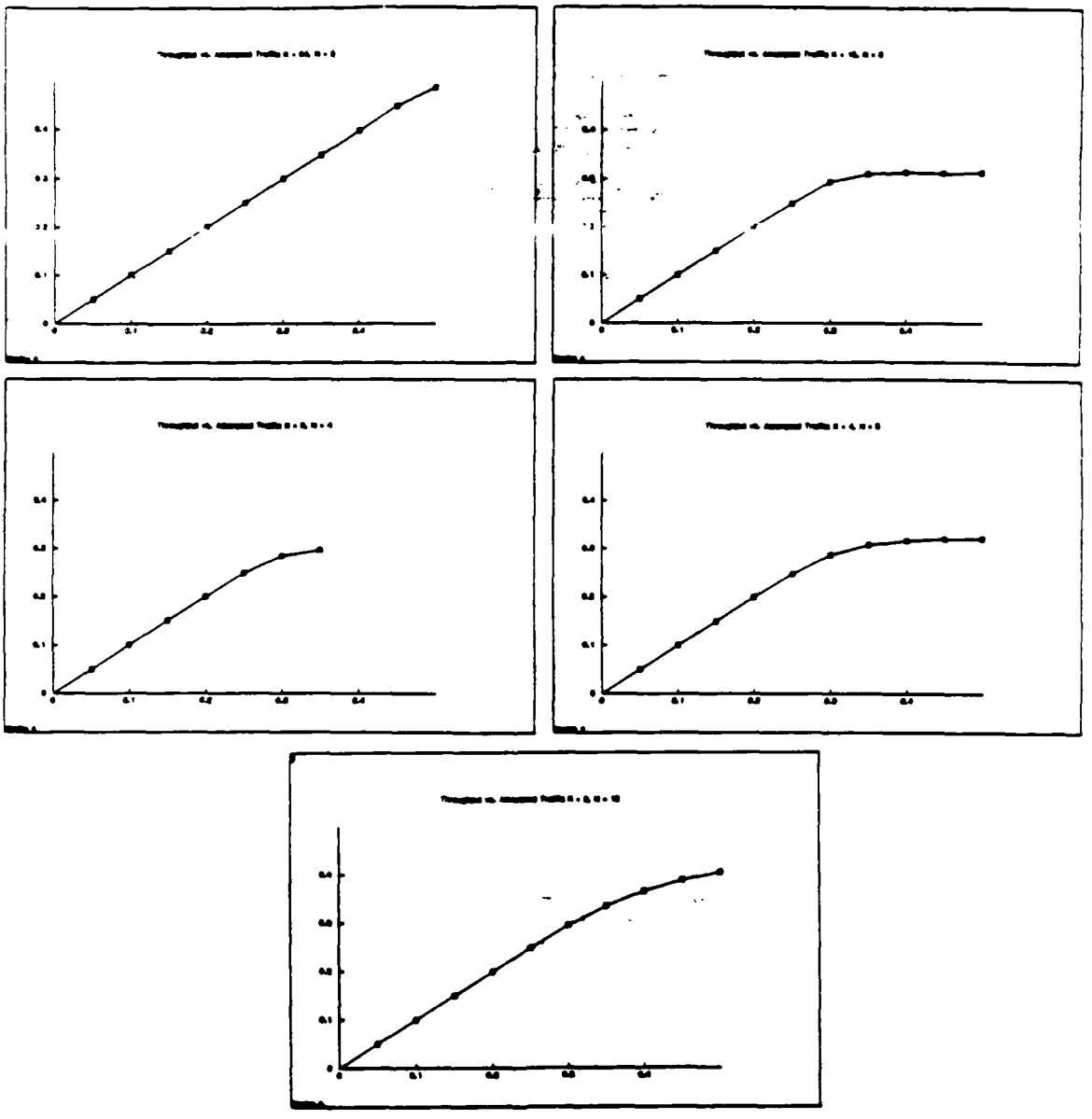


Figure 17: Actual Traffic vs. Attempted Traffic for 4K node networks: 64-ary 2-cube, 16-ary 3-cube, 8-ary 4-cube, 4-ary 6-cube, and binary 12-cube, L=200bits.

| Parameter | 1K Nodes | | | 4K Nodes | | | | |
|---|---|---|---|---|---|---|---|---|
| Dimension | 2 | 5 | 10 | 2 | 3 | 4 | 6 | 12 |
| radix | 32 | 4 | 2 | 64 | 16 | 8 | 4 | 2 |
| Max Throughput | 0.36 | 0.41 | 0.43 | 0.35 | 0.31 | 0.31 | 0.36 | 0.41 |
| Latency $\lambda = 0.1$ | 46.1 | 128. | 233. | 70.7 | 55.2 | 79.9 | 135. | 241. |
| Latency $\lambda = 0.2$ | 50.5 | 161. | 269. | 73.1 | 70.3 | 112. | 181. | 288. |
| Latency $\lambda = 0.3$ | 59.3 | 221. | 317. | 78.6 | 135. | 245. | 287. | 357. |

Table 1: Maximum Throughput as a Fraction of Capacity and Blocking Latency in Cycles

When the network saturates, throughput levels off as shown in Figures 16 and 17. These figures show how much traffic is delivered (vertical axis) when the nodes attempt to inject a given amount of traffic (horizontal axis). The curve is linear (actual = attempted) until saturation is reached. From this point on, actual traffic is constant. This plateau occurs because (1) the network is source queued, and (2) messages that encounter contention are blocked rather than aborted. In networks where contention is resolved by dropping messages, throughput usually decreases beyond saturation.

To find the maximum throughput of the network, the source service time, $T_0$, is set equal to the reciprocal of the message rate, $\lambda_E$, and equations (19), (20), and (21) are solved for $\lambda_E$. The maximum throughput as a fraction of capacity for $k$-ary $n$-cubes with 1K and 4K nodes is tabulated in Table 1. Also shown is the total latency for L = 200bit messages at several message rates. The table shows that the additional latency due to blocking is significantly reduced as dimension is decreased.

In networks of constant bisection width, the latency of low-dimensional networks increases more slowly with applied traffic than the latency of high-dimensional networks. At $\lambda = 0.2$, the 32-ary 2-cube has $\approx \frac{1}{5}$ the latency of the binary 10-cube. At this point, the additional latency due to contention in the 32-ary 2-cube is $7T_c$ compared to $64T_c$ in the binary 10-cube. At moderate loads, low-dimensional networks may outperform higher-dimensional networks with lower zero-load latency. For example, a 16-ary 3-cube has lower zero-load latency than a 64-ary 2-cube (47.5 vs. 69.25). However, the 64-ary 2-cube has lower latency when $\lambda = 0.3$ (78.6 vs 135).

Intuitively, low-dimensional networks handle contention better because they use fewer channels of higher bandwidth and thus get better queueing performance. The shorter service times, $\frac{1}{W}$, of these networks results in both a lower probability of collision, and a lower expected waiting time in the event of a collision. Thus the blocking latency at each node is reduced quadratically as $k$ is increased. Low-dimensional networks require more hops, $D = \frac{n(k-1)}{2}$, and have a higher rate on the continuing channels, $\lambda_C$. However, messages travel on the continuing channels more frequently than on the entering channels, thus most contention is with the lower rate channels. Having fewer channels of higher bandwidth also improves hot-spot throughput as described below.

### 3.3 Hot Spot Throughput

In many situations traffic is not uniform, but rather is concentrated into *hot spots*. A *hot spot* is a pair of nodes that accounts for a disproportionately large portion of the total network traffic. As described by Pfister [16] for a shared-memory computer, hot-spot traffic can degrade performance of the entire network by causing congestion.

The *hot-spot throughput* of a network is the maximum rate at which messages can be sent from one specific node, $P_i$, to another specific node, $P_j$. For a $k$-ary $n$-cube with deterministic routing, the hot-spot throughput, $\Theta_{HS}$, is just the bandwidth of a single channel, $W$. Thus, under the assumption of constant wire cost we have

$$\Theta_{HS} = W = k - 1. \tag{24}$$

Low-dimensional networks have greater channel bandwidth and thus have greater hot-spot throughput than do high-dimensional networks. Intuitively, low-dimensional networks operate better under non-uniform loads because they do more resource sharing. In an interconnection network the resources are wires. In a high-dimensional network, wires are assigned to particular dimensions and cannot be shared between dimensions. For example, in a binary $n$-cube it is possible for a wire to be saturated while a physically adjacent wire assigned to a different dimension remains idle. In a torus all physically adjacent wires are combined into a single channel that is shared by all messages that must traverse the physical distance spanned by the channel.

## 4 Conclusion

Under the assumption of constant wire bisection, low-dimensional networks with wide channels provide lower latency, less contention, and higher hot-spot throughput than high-dimensional networks with narrow channels. Minimum network latency is achieved when the network radix, $k$, and dimension, $n$, are chosen to make the components of latency due to distance, $D$, and aspect ratio, $\frac{k}{W}$ approximately equal. The minimum latency occurs at a very low dimension, 2 for up to 1024 nodes.

Low dimensional networks reduce contention because having a few high-bandwidth channels results in more resource sharing and thus better queueing performance than having many low-bandwidth channels. While network capacity and worst-case blocking latency are independent of dimension, low-dimensional networks have a higher maximum throughput and lower average blocking latency than do high-dimensional networks. Improved resource sharing also gives low-dimensional networks higher hot-spot throughput than high-dimensional networks.

The results of this paper have all been made under the assumption of constant channel delay, independent of channel length. The main result, that low-dimensional networks give minimum latency, however, does not change appreciably when logarithmic or linear delay models are considered. In choosing a delay model one must consider how the delay of a switching node

compares to the delay of a wire. Current VLSI routing chips [7] have delays of tens of nanoseconds, enough time to drive several meters of wire. For such systems a constant delay model is adequate. As chips get faster and systems get larger, however, a linear delay model will more accurately reflect system performance.

Fat-tree networks have been shown to be universal in the sense that they can *efficiently* simulate any other network of the same volume [13]. However, the analysis of these networks has not considered latency. $k$-ary $n$-cubes with appropriately chosen radix and dimension are also universal in this sense. A detailed proof is beyond the scope of this paper. Intuitively, one cannot do any better than to fill each of the three physical dimensions with wires and place switches at every point of intersection. Any point-to-point network can be embedded into such a 3-D mesh with no more than a constant increase in wiring length.

This paper has considered only *direct* networks [17]. The results do not apply to *indirect* networks. The depth and the switch degree of an indirect network are analogous to the dimension and radix of a direct network. However, the bisection width of an indirect network is independent of switch degree. Because indirect networks do not exploit locality it is not possible to trade off diameter for bandwidth. There is little reason to construct an indirect network. A high-bandwidth direct network would provide the same function with increased performance.

The low-dimensional $k$-ary $n$-cube provide a very general communication media for digital systems. These networks have been developed primarily for message-passing concurrent computers. They could also be used in place of a bus or indirect network in a shared-memory concurrent computer, in place of a bus to connect the components of a sequential computer, or to connect subsystems of a special purpose digital system. With VLSI communication chips the cost of implementing a network node is comparable to the cost of interfacing to a shared bus, and the performance of the network is considerably greater than the performance of a bus.

The Torus Routing Chip (TRC) is a VLSI chip designed to implement low-dimensional $k$-ary $n$-cube interconnection networks [7]. The TRC performs wormhole routing in arbitrary $k$-ary $n$-cube interconnection networks. This self-timed chip was functional on first silicon. A single TRC provides 8-bit data channels in two dimensions and can be cascaded to add more dimensions or wider data channels. A TRC network can deliver a 150-bit message in a 1024 node 32-ary 2-cube with an average latency of $7.5\mu s$, an order of magnitude better performance than would be achieved by a binary $n$-cube with bit-serial channels. A new routing chip, the Network Design Frame (NDF), currently under development, is expected to improve this latency to $\approx 1\mu s$. [10]

Now that the latency of communication networks has been reduced to a few microseconds the latency of the processing nodes, $T_{node}$, dominates the overall latency. To efficiently make use of a low-latency communication network we need a processing node that interprets messages with very little overhead. The design of such a *message-driven processor* is currently underway [5] [9].

The real challenge in concurrent computing is software. The development of concurrent software is strongly influenced by available concurrent hardware. We hope that by providing machines with higher performance internode communication we will encourage concurrency to

be exploited at a finer grain size in both system and application software.

## Acknowledgments

I thank Chuck Seitz of Caltech for his many helpful suggestions during the early stages of this research.

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency in part under contract number N00014-80-C-0622 and in part under contract number N00014-85-K0124.

## References

[1] Batcher, K.E., "Sorting Networks and Their Applications," *Proceedings AFIPS FJCC*, Vol. 32, 1968, pp. 307-314.

[2] Batcher, K.E., "The Flip Network in STARAN," *Proceedings, 1976 International Conference on Parallel Processing*, pp. 65-71.

[3] Benes, V.E., *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic, New York, 1965.

[4] Browning, Sally, *The Tree Machine: A Highly Concurrent Computing Environment*, Dept. of Computer Science, California Institute of Technology, Technical Report 3760, 1985.

[5] Dally, William J., *A VLSI Architecture for Concurrent Data Structures*, Kluwer, Hingham, MA, 1987.

[6] Dally, William J. and Seitz, Charles L., *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks, IEEE Transactions on Computers*, Vol. C-36, No. 5, May 1987, pp. 547-553.

[7] Dally, William J. and Seitz, Charles L., "The Torus Routing Chip," *J. Distributed Systems*, Vol. 1, No. 3, 1986, pp. 187-196.

[8] Dally, William J. "Wire Efficient VLSI Multiprocessor Communication Networks," *Proceedings Stanford Conference on Advanced Research in VLSI*, Paul Losleben, Ed., MIT Press, Cambridge, MA, March 1987, pp. 391-415.

[9] Dally, William J., "Architecture of a Message-Driven Processor," *Proceedings of the* 14$^{th}$ *ACM/IEEE Symposium on Computer Architecture*, June 1987, pp. 189-196..

[10] Dally, William J., and Song, Paul., "Design of a Self-Timed VLSI Multicomputer Communication Controller," To appear in, *Proc. IEEE International Conference on Computer Design*, 1987.

[11] Kermani, Parvis and Kleinrock, Leonard, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, Vol 3., 1979, pp. 267-286.

[12] Lawrie, Duncan H., "Alignment and Access of Data in an Array Processor," *IEEE Transactions on Computers*, Vol. C-24, No. 12, December 1975, pp. 1145-1155.

[13] Leiserson, Charles, L., "Fat Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers*, Vol. C-34, No. 10, October 1985, pp. 892-901.

[14] Mead, Carver A. and Conway, Lynn A., *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass., 1980.

[15] Pease, M.C., III, "The Indirect Binary n-Cube Microprocessor Array," *IEEE Transactions on Computers*, Vol. C-26, No. 5, May 1977, pp. 458-473.

[16] Pfister, G.F. and Norton, V.A., "Hot Spot Contention and Combining in Multistage Interconnection Networks," *IEEE Transactions on Computers*, Vol. C-34, No. 10, October 1985, pp. 943-948.

[17] Seitz, Charles L., "Concurrent VLSI Architectures," *IEEE Transactions on Computers*, Vol. C-33, No. 12, December 1984, pp. 1247-1265.

[18] Seitz, Charles L., et al., *The Hypercube Communications Chip*, Dept. of Computer Science, California Institute of Technology, Display File 5182:DF:85, March 1985.

[19] Sequin, Carlo, H., "Single Chip Computers, The New VLSI Building Block," *Caltech Conference on VLSI*, C. L. Seitz Ed. January 1979, pp. 435-452.

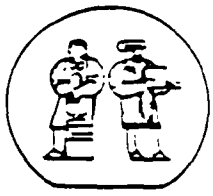
[20] Siegel, Howard Jay, "Interconnection Networks for SIMD Machines," *IEEE Computer*, Vol. 12, No. 6, June 1979, pp. 57-65.

[21] Stone, H.S., "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers*, Vol. C-20, No. 2, February 1971, pp. 153-161.

[22] Sullivan, H. and Bashkow, T.R., "A Large Scale Homogeneous Machine," *Proc. 4th Annual Symposium on Computer Architecture*, 1977, pp. 105-124.

[23] Tanenbaum, A. S., *Computer Networks*, Prentice Hall, Englewood Cliffs, N.J., 1981.

[24] Thompson, C.D., *A Complexity Theory of VLSI*, Department of Computer Science, Carnegie-Mellon University, Technical Report CMU-CS-80-140, August 1980.





VLSI Memo No. 87-426
November 1987

# SPACE-EFFICIENT QUEUE MANAGEMENT USING FIXED-CONNECTION NETWORKS

Tom Leighton and Eric Schwabe

Abstract

One of the main difficulties in designing algorithms for large scale parallel machines is making sure that the capacities of the local memories are not exceeded. In this paper, we present a general scheme for dynamically reorganizing memory so that local memory constraints are never exceeded provided that global memory constraints are not exceeded. The scheme is simple, real-time, space-efficient, deterministic and transparent to the programmer. It requires only that the total hardware used (i.e., wires and total memory) exceed the number of local memories by a logarithmic factor. In return, the scheme guarantees an arbitrarily high percentage utilization of the total memory, independent of whatever local demands for memory arise. We analyze the behaviour of our scheme in worst-case and average-case settings, and we show that it is optimal in many respects, even when compared to randomized algorithms.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138

# Space-Efficient Queue Management Using Fixed-Connection Networks

Tom Leighton*
Eric Schwabe†
MIT Department of Mathematics
and Laboratory for Computer Science
Cambridge, MA 02139

**Abstract**

One of the main difficulties in designing algorithms for large scale parallel machines is making sure that the capacities of the local memories are not exceeded. In this paper, we present a general scheme for dynamically reorganizing memory so that local memory constraints are never exceeded provided that global memory constraints are not exceeded. The scheme is simple, real-time, space-efficient, deterministic and transparent to the programmer. It requires only that the total hardware used (i.e., wires and total memory) exceed the number of local memories by a logarithmic factor. In return, the scheme guarantees an arbitrarily high percentage utilization of the total memory, independent of whatever local demands for memory arise. We analyze the behaviour of our scheme in worst-case and average-case settings, and we show that it is optimal in many respects, even when compared to randomized algorithms.

## 1 Introduction

In very large scale parallel computers, memory is distributed in small chunks among a large number of processors. Typically, the total memory of the system is quite large, but each local memory is quite small. Insuring that the capacity of the local memories is not exceeded is one of the main problems in designing parallel algorithms and architectures. Indeed, it is often a relatively simple matter to insure that total memory capacity is not exceeded, but fluctuations in the demand for local resources invariably arise that make allocation of memory in certain "hot spots" much more difficult to handle.

As an example, consider the problem of routing a permutation of $N$ packets on an $N$-processor machine. At the beginning and the end of the routing, each processor has precisely

*supported by Air Force Contract OSR-86-0076, DARPA Contract N00014-80-C-0622, the MIT Army Center for Intelligent Control, and an NSF Presidential Young Investigator Award with matching funds from ATT and IBM

†supported by a National Science Foundation Graduate Fellowship

one packet. Depending on the algorithm used to route the packets, however, large numbers of packets might bunch up at a few nodes during intermediate stages of the routing. For example, up to $\Omega(\sqrt{N})$ packets might bunch up at a single node if an oblivious routing strategy is used [BH]. Although we could handle the local space crunch by allocating $\Theta(\sqrt{N})$ capacity to each local memory, it would be terribly wasteful to do so since we would only be using a total of $N$ out of $\Theta(N^{3/2})$ space overall at any point in time.

A variety of approaches are used in practice to try to overcome the problems associated with localization of memory. These approaches include: allocation of "extra" space at each processor, misallocation (i.e., sending data where it doesn't want to go, but where there is space), destruction of overflow data, freezing movement of data until space becomes available, and randomizing the desired location of data by (for example) hashing the memory on a global basis. Each of these techniques has its advantages and disadvantages. Probably the most popular is randomization which has been applied quite successfully to packet routing problems [VB] [R], but even here there is no iron-clad guarantee of success, and operations such as hashing the entire memory involve a great deal of overhead. Moreover, randomization has only been shown to work for certain very specific problems.

In this paper, we present a general scheme for automatically reorganizing memory so that local memory bounds are never exceeded provided that global memory bounds are not exceeded. The scheme is real-time, space-efficient, deterministic, transparent to the programmer, and perhaps best of all, quite simple. It requires only that the total hardware used (i.e., wires and total memory) exceed the number of local memories being simulated by a logarithmic factor. In return, the scheme guarantees an arbitrarily high percentage utilization of the total memory, independent of whatever local demands for memory arise.

Our model assumes that each local memory can be treated as a linear array that is accessed through a *port* associated with a single processor in the network. This is somewhat restrictive in that we do not allow a processor to have random access to its local memory, but it is general enough to include memories that are queues, stacks, priority queues, and the like. We break up memory and data into fundamental units called *packets*. We assume that one packet can traverse a single wire in a single unit of time. Packets can be thought of as bits, bytes, or entire blocks of data. The arrival and departure of data packets through a port is governed by a parameter $\lambda$ that is usually assumed to be a constant. In particular, we assume that no more than $\lceil \lambda t \rceil$ packets can arrive or depart through a port in $t$ consecutive steps. Lastly, we let $m$ denote the number of local memories, and $p$ denote the total number of packets in all the memories. Note that the $p$ packets may or may not be distributed evenly among the $m$ processors. Indeed, all $p$ packets may be contained in a single local memory.

The task facing us is to design a network and algorithm to simulate any action of $m$ local memories subject to the preceding constraints. To effect the simulation, we will construct an $N$-node degree-$d$ network where each processor has a local memory of size $q$. Of course, if $q$ is as large as $p$, the simulation is trivial. The object is to make $N$, $d$ and $q$ as small as possible relative to $m$, $\lambda$ and $p$. In this paper, we describe a butterfly-based construction for which $\lambda = \Theta(1)$, $d = \Theta(1)$, $q = \Theta(1)$ and $N = \max(p, m \log m)$, and a hypercube-based construction for which $\lambda = \Theta(1)$, $d = \Theta(\log N)$, $q = \Theta(\log N)$ and $N = \max(m, p/\log m)$.

Both constructions are optimal in the sense that the total storage needed for the simulation ($Nq$) is only slightly larger than the trivial lower bound of $p$ when $p$ is large, and in the sense that the simulations are real time. The butterfly-based construction is also optimal in the sense that any bounded-degree simulator must have $\Omega(m \log m)$ total memory, even if $p$ is small and even if randomized algorithms are used. Whether or not a simulator can be constructed with fewer nodes and wires (say $m$) but with more local memory per node (say $\log m$) when $p = m \log m$ remains an open question. We show that such a simulator does exist when $p = m$, but this result is less interesting since it is not space-efficient.

Our simulations are based on an efficient solution to a specialized routing problem that we call *isotone routing*. In particular, an *isotone routing problem* is one for which the relative order of the elements being moved is unchanged. For example, the mapping $\{1 \rightarrow 3, 2 \rightarrow 4, 6 \rightarrow 5, 7 \rightarrow 8\}$ is an *isotone partial permutation*. We suspect isotone routing problems will turn out to be useful in other applications as well. As a simple example, we show how to use our isotone routing algorithm to route any permutation problem of size $N$ on an $N \log N$-node butterfly in $O(\log^2 / \log \log N)$ steps, slightly improving the previous best known deterministic bound of $\Theta(\log^2 N)$

The remainder of the paper is divided into four sections. Our memory reallocation scheme and related constructions are presented in Section 2. Section 3 contains the lower bounds and proofs of optimality. Average-case results are dicussed in Section 4. Here it is interesting to note that although randomized algorithms fare no better than deterministic algorithms on worst-case problems, an average-case problem is substantially easier to handle than a worst-case problem. We conclude with some open questions and subjects of ongoing research in Section 5.

Due to the constraints on length, proofs and descriptions of constructions are severely abbreviated. We will also restrict our attention to the special case when each local memory is a simple queue.

# 2 Constructions

## 2.1 Isotone Routing

The building block of the space-efficient queueing networks discussed in this section is a butterfly-based network which can route a special class of partial permutations of $\{1, 2, \ldots, m\}$ deterministically and on-line in $\Theta(\log m)$ steps. The network in Figure 1 (for $m = 8$) can perform such routing for *isotone* partial permutations; we say that a partial permutation of $\{1, 2, \ldots, m\}$ is isotone if the mapping from sources to destinations is strictly isotone (since we are dealing with a partial permutation, this mapping is injective, so that the mapping being isotone is equivalent to its being strictly isotone). Note that the middle row of columns is not really necessary (it is included to simplify the algorithm description), and that this network, aside from the long vertical edges, is just a column permutation of the Benes back-to-back butterfly network.

The network can route an isotone partial permutation of $\{1, 2, \ldots, m\}$ as follows. The upper butterfly is used to count the sources, and to assign to each source its position in the left-to-right ordering of sources (0 for the leftmost source, 1 for the next...): this can be done using a prefix calculation, and an extra step to pass the values back to the top — a total of $\log m + 1$ steps. Then in $2 \log m$ steps, each source can be routed to the position in the middle row determined by this value and then on to its destination without collisions. That there are no collisions comes from the fact that, since we are packing all the sources to the left, we are in effect performing two instances (one in reverse) of deterministic on-line 0-1 routing (given 0's and 1's distributed among the sources, route them so that all the 0's are to the left of all the 1's), for which it is known that no collisions occur for routing using straightforward bit-flipping. Thus the isotone partial permutation is routed in $3 \log m + 1$ steps.

Note that if the sources are the same for a sequence of $j$ isotone partial permutations (or even if each set of sources is included in the previous set), then we can use the results of the first prefix computation to route all $j$ permutations one after the other, for a total time of $(\log m + 1) + (2 \log m + j - 1) = j + 3 \log m$. In particular, when each member of the set of sources $s_1 < s_2 < \ldots < s_i$ has some discrete interval of destinations of length at most $j$ ($\{d_i, d_i + 1, \ldots, d_i + j_i - 1\}$, $j_i \leq j$), where for all $k$, $d_k + j_k - 1 < d_{k+1,1}$ (all the destinations of $s_k$ are less than all the destinations of $s_{k+1}$), the above conditions are satisfied, and the routing can be accomplished in $j + 3 \log m$ steps.

Note also that by 'folding up' the two butterflies in the network, and identifying the resulting top and bottom rows of processors, we could reduce the size of the network to $m \log m$, and allow it to perform isotone routing in $3 \log m$ steps. However, this would eliminate any possibility of pipelining as described above, making the current configuration a better choice for the queueing algorithm, which relies heavily on such pipelining. Also, adding another set of long edges across the lower butterfly will allow us perform such routing in both directions on the network.

## 2.2 Queueing Networks and Algorithms

The queue management problem can be solved for $q = \Theta(1)$, $N = \Theta(m \log m)$, $\lambda < 1$ and $p$ an arbitrarily large constant fraction of $qN$ on a butterfly-based network.

In its most basic form, this network consists of $m$ linear arrays of processors of size $C \log m$ (discussion of the $C$ and the other constants involved in the construction, which affect the values of $\lambda$ for which the algorithm will be correct, is omitted here), each with one of the $m$ ports at one end, and at the other end a connection to a bidirectional isotone routing network. At the bottom of the isotone routing network are $m$ linear arrays of size $\log m$. For example, see Figure 2.

The idea is that each of the $m$ queues has some number (between $(u - v) \log m$ and $(u + v + 1) \log m$) of its elements stored in its upper linear array, and the excess stored in some range of the lower linear arrays. The algorithm cycles repeatedly, each time adjusting the number of elements in each upper linear array by either sending some multiple of $\log m$ queue

elements down to the lower linear arrays, or bringing up some multiple of $\log m$ elements from the lower linear arrays.

Each cycle consists of four phases (presented here in order of execution: in an enhanced version of the network and algorithm, further parallelism is used to sharpen results):

1. Overhead, in which it is determined for each processor whether queue elements will be sent down or retrieved, and its new range of lower linear arrays is calculated.

2. Retrievals, where all queues which have become 'too small' simultaneously bring elements from their ranges of lower linear arrays back to their upper linear arrays, using isotone routing.

3. Shifts, where those elements remaining in the lower linear arrays are moved to their new positions (as calculated in the Overhead phase), in preparation for new elements to be sent down.

4. Sends, where all queues which have become 'too big' simultaneously send elements form their upper linear arrays down to their ranges of lower linear arrays.

Analysis of the time required by these cycles of the algorithm gives us constraints on the values of $\lambda$ for which the algorithm will work in terms of $u, v$, and $C$, and allows us to find the optimal values of $u$ and $C$ for a given $v$. For the basic network and algorithm, we will always have $\lambda < \frac{1}{2}$, but we also describe an enhanced version with a higher degree of parallelism which allows us to get $\lambda$ arbitrarily close to 1 by choosing sufficiently large $v$. Although to simplify the explanation, this algorithm is explained in terms of managing stacks, adjustments can be made to allow us to simulate general linear arrays (i.e., stacks, priority queues, FIFO, etc.).

We also describe a hypercube-based algorithm, obtained by identifying entire columns of the butterfly-based network into single nodes. This network solves the problem with $q = \Theta(\log m)$ and $N = m$.

In their paper on the token distribution problem, Peleg and Upfal describe an $n$-node bounded-degree network which can, with $\Theta(\log n)$ local storage, solve routing problems for up to $n$ packets in $\Theta(\log n)$ time, as long as no source or destination node has multliplicity greater than $\log n$. One might try replacing the isotone routing network with an $m$-node network of this type and transforming each sub-array of $\log m$ nodes in the linear arrays into a single node with $\log m$ size internal storage, yielding a bounded-degree network with $q = \Theta(\log m)$ and $N = \Theta(m)$ to solve the queueing problem. However, since the routing network has no pipelining abilities, we would have to restrict the total number of packets in the network to be $N = m$, only using $\frac{1}{\log m}$ of the available space.

## 2.3 An Application of Isotone Routing to General Permutation Routing

There is a deterministic algorithm for routing a partial permutation on an $n$-node hypercube in $\Theta(\frac{\log^2 n}{\log \log n})$ steps [K] [L] which, using isotone routing, yields a partial permutation routing algorithm on the butterfly which runs in the same number of steps.

The hypercube algorithm goes as follows: Route greedily for the first $\log\log n$ bits. At this point, each $\log n$-node subcube defined by a setting of these $\log\log n$ bits can contain at most $\log n$ packets. Using isotone routing we can spread out these packets in their respective subcubes, at most one to each node, in $\Theta(\log n)$ steps; repeat this process for each subproblem. Thus the total running time is $\frac{\log n}{\log\log n}\Theta(\log n) = \Theta(\frac{\log^2 n}{\log\log n})$.

On the butterfly, the corresponding algorithm is as follows: route greedily for the first $\log\log n$ levels of the butterfly; then, as before, each sub-butterfly defined by a setting of these $\log\log n$ bits can contain at most $\log n$ packets. Thus we can use isotone routing to distribute these packets evenly within the sub-butterfly in $\Theta(\log n)$ steps; repeat this process on each sub-butterfly. As before, the total running time is $\Theta(\frac{\log^2 n}{\log\log n})$. The key difference is that we must simulate the $\log n$-size queues of the hypercube with constant-size queues contained in the corresponding nodes of the butterfly. The details are not difficult to work out.

# 3 Lower Bounds

Here we prove lower bounds on the space needed to solve the memory management problem, for both deterministic and randomized algorithms. In looking for lower bounds, we consider an *off-line* formulation of the problem where we are given a final queue length for each port in the network and the problem is to find $m$ slot-disjoint queues of the appropriate lengths from the $m$ ports in the network; nodes of the network have random access to their slots. (A *slot* is a single location in a local memory of the simulating network.) Since the general queueing problem is at least as hard as the off-line problem, any lower bounds found for the off-line problem will apply to the on-line problem.

It is immediate from the problem definition that $m \leq N$ and $p \leq qN$; also, we must have $q \geq \lceil\lambda\rceil \geq \lambda$, since there must be room in a port to hold all the packets which could arrive (or be requested) in one time step. In addition, for $r$ = the length of the longest queue and $T$ = the number of time steps, we must have $r \leq \lceil\lambda T\rceil$.

## 3.1 The Deterministic Case

**Lemma 1D:** Let a queue-finding algorithm on an $N$-node network with $m$ ports be given. If $mr > qN$ and $6rd^T \leq p$, then there is some assignment of queue lengths to the $m$ ports for which the algorithm will not find disjoint queues of the appropriate lengths in the network.

**Sketch of Proof:** Assume that $mr > qN$ and $6rd^T \leq p$. If these constraints are satisfied, then we can force two ports to select queues which intersect in some slot as follows:

Since the total number of nodes at distance no greater than $T$ from a given node is at most $3d^T$, each port's choice of queue can depend on the values in at most $3d^T$ ports. For each of the $m$ ports, consider the queue of length $r$ chosen when all $3d^T$ of the ports within distance $T$ are assigned the value $r$. The total length of these queues is $mr$; since $mr > qN$, we can choose two ports for which these queues intersect at some slot. We assign values to ports as follows: assign to the two ports chosen above and to each port within distance $T$ of either of them the value $r$; this is permissible since it involves at most $6d^T$ ports, and $6rd^T \leq p$ (there

are enough packets to go around). Assign any extra packets arbitrarily. This assignment of values to ports will cause the algorithm to fail by choosing intersecting queues, thus proving the Lemma.

**Lemma 2D:** If $4qN \log d < \lambda m \log \frac{mp}{12qN}$, then there is no correct queue-finding algorithm.

**Sketch of Proof:** Assume that $4qN \log d < \lambda m \log \frac{mp}{12qN}$, and let a queue-finding algorithm be given. Then it can be shown $r = \lfloor \frac{qN}{m} + 1 \rfloor$ and $T = \lfloor \frac{\log \frac{mp}{12qN}}{\log d} \rfloor$ satisfy $r \leq \lceil \lambda T \rceil$ and the conditions of Lemma 1D. It follows that the queue-finding algorithm must fail on some input. Therefore no correct queue-finding algorithm exists.

The following theorem shows a lower bound on the size of a family of $N = N(m)$-node networks with $m$ ports, with a total queue length of $p = p(m)$.

**Theorem:** Let $N = N(m)$, and $p = p(m) \geq (qN(m))^{\epsilon}$ for some $\epsilon \in (0, 1]$ and sufficiently large $m$. If $qN \log d = o(m \log m)$ then no family of $N$-node networks with correct queue-finding algorithms for $p$ packets exists.

**Sketch of Proof:** Assume that the above conditions hold, and let a family of $N$-node networks be given. Suppose that this family had correct queue-finding algorithms. Since $qN \log d = o(m \log m)$, we have that for all $c > 0$ and for sufficiently large $m$, $qN \log d < cm \log m$. Choose $c_1 > 0$ such that for sufficiently large $m$, $c_1 \leq \frac{\lambda}{4}(\epsilon + (\epsilon - 1)\frac{\log \log m}{\log m})$, and choose $c_2 > 0$ such that $c_2 \leq 12^{\frac{1}{\epsilon - 1}}$ (if $\epsilon = 1$, choose any $c_2 > 0$).

For these constants and sufficiently large $m$, $4qN \log d < \lambda m \log \frac{mp}{12qN}$, implying that (by Lemma 2D) no correct queue-finding algorithm can exist for sufficiently large $m$. This contradicts that this is a family of networks with correct queue-finding algorithms. We conclude that no such family exists.

## 3.2 The Randomized Case

For the randomized lower bound we use the most general version of an off-line randomized algorithm, where there are an arbitrary number of public random bits which all processors can access and, as before, each port has a value which is its final queue length.

**Claim:** Suppose $mr > 3qN$. Then for any set of $r$-length queues from the $m$ ports, there exist distinct ports $p_1, p_2, \ldots, p_{\frac{2m}{3}}$ such that for all $i \in 1, 2, \ldots, \frac{m}{3}$, the queues from $p_{2i-1}$ and $p_{2i}$ intersect in some slot.

**Sketch of Proof:** Since $mr > 3qN$, we have that $[m - 2(i-1)]r > qN$ for $i \in 1, 2, \ldots, \frac{m}{3}$. Thus for $i = 1, 2, \ldots, \frac{m}{3}$ we can find an intersecting pair of queues, and remove them and their ports from consideration. This will yield $\frac{m}{3}$ distinct pairs of ports whose queues intersect, as desired.

**Claim:** Suppose $mr > 3qN$. Then given any randomized queue-finding algorithm (and sufficiently large $m$), there is some set of $2m^{\frac{1}{2}+\delta}$ ports (for any $\delta \in (0, \frac{1}{2})$) for which, when the values of all ports looked at during the computation are $r$, we have that Pr(the algorithm fails for bit setting $B$) is exponentially close to 1.

**Sketch of Proof:** Suppose $mr > 3qN$, and let a randomized queue-finding algorithm and a setting of the random bits $B$ be given.

We consider the collision graph $G$, whose nodes are the $m$ ports, and for which there is an edge between $x$ and $y$ in $G$ if and only if the queues from ports $x$ and $y$ intersect when all values seen in other ports are $r$, and the random bits are set according to $B$. By the previous Claim, there is a set of $\frac{m}{3}$ independent edges (edges with no common endpoints) in $G$.

We then show that for a random set of $2m^{\frac{1}{2}+\delta}$ ports, the probability that the induced subgraph of the collision graph contains at least one of these edges is exponentially close to 1. This means that for almost all sets of $2m^{\frac{1}{2}+\delta}$ ports, if all the ports looked at by any of these ports had value $r$, then the algorithm would choose some pair of intersecting queues for bit setting $B$. Therefore for each possible bit setting $B$, all but exponentially few of the possible sets of $2m^{\frac{1}{2}+\delta}$ ports cover an edge in the collision graph, so that there must be some set of $2m^{\frac{1}{2}+\delta}$ ports which covers an edge in the collision graph for all but exponentially few bit settings. This is the set required by the Claim. In fact, the same property holds for all but exponentially few of the sets of $2m^{frac12+\delta}$ ports.

From this point, the argument follows the same program as the proof of the deterministic lower bound.

**Lemma 1R:** Let a randomized queue-finding algorithm on an $N$-node network with $m$ ports be given. If $mr > 3qN$ and $6m^{\frac{1}{2}+\delta}rd^T \leq p$, then there is some assignment of queue lengths to the $m$ ports for which the algorithm will fail (that is, not find disjoint queues of the appropriate lengths in the network) with probability exponentially close to 1.

**Sketch of Proof:** Assume that $mr > 3qN$ and $6m^{\frac{1}{2}+\delta}rd^T \leq p$. Consider a set of $2m^{\frac{1}{2}+\delta}$ for which all but exponentially few of the possible settings of the random bits cause the algorithm to select intersecting queues when the values in all the ports looked at during the computation are $r$. As in the determinisitc argument, a port can learn the values in at most $3d^T$ ports in $T$ steps.

Thus by assigning the value $r$ to the $6m^{\frac{1}{2}+\delta}d^T$ ports which can be looked at during the course of the computation (and assigning any extra packets arbitrarily), we insure that for all but exponentially few of the possible settings of the random bits intersecting queues will be chosen. Thus for this assignment of values to ports, the algorithm will fail with probability exponentially close to 1.

**Lemma 2R:** If $12qN \log d < \lambda m \log \frac{m^{\frac{1}{2}-\delta}p}{36qN}$, then there is no randomized queue-finding algorithm for which we cannot find an input on which it fails with probability exponentially close to 1.

**Sketch of Proof:** Assume that $12qN \log d < \lambda m \log \frac{m^{\frac{1}{2}-\delta}p}{36qN}$, and let a randomized queue-finding algorithm be given. Then $r = \lfloor \frac{3qN}{m} + 1 \rfloor$ and $T = \lfloor \frac{\log \frac{m^{\frac{1}{2}-\delta}p}{36qN}}{\log d} \rfloor$ satisfy $r \leq \lceil \lambda T \rceil$ and the conditions of Lemma 1R. It follows that the randomized queue-finding algorithm must on some input almost certainly fail. Therefore no randomized queue-finding algorithm for which such an input cannot be found can exist.

The following theorem shows a lower bound on the size of a family of $\mathcal{N} = \mathcal{N}(m)$-node networks with $m$ ports, with a total queue length of $p = p(m)$, which is within a constant factor of the lower bound already found for deterministic algorithms.

**Theorem:** Let $\mathcal{N} = \mathcal{N}(m)$, and $p = p(m) \geq (qN(m))^{\epsilon+\frac{1}{2}}$ for some $\epsilon \in (0, \frac{1}{2}]$ and sufficiently large $m$. If $q\mathcal{N} \log d = o(m \log m)$ then no family of $N$-node networks with randomized queue-finding algorithms for $p$ packets where we cannot find some bad input exists.

**Sketch of Proof:** Assume that the above conditions hold, and let a family of $N$-node networks be given. Suppose that this family had randomized queue-finding algorithms without bad inputs. Choose $\delta \in (0, \epsilon)$. Since $qN \log d = o(m \log m)$, we have that for all $c > 0$ and for sufficiently large $m$, $qN < cm \log m$. Choose $c_1 > 0$ such that for sufficiently large $m$, $c_1 \leq \frac{\lambda}{12}(\epsilon - \delta + (\epsilon - \frac{1}{2})\frac{\log\log m}{\log m})$; this quantity is positive for sufficiently large $m$, since $\epsilon > \delta$). Choose $c_2 > 0$ such that $c_2 \leq 12^{\frac{1}{\epsilon-\frac{1}{2}}}$ (if $\epsilon = \frac{1}{2}$, choose any $c_2 > 0$).

For these constants and sufficiently large $m$, $12qN \log d < \lambda m \log \frac{m^{\frac{1}{2}-\delta}p}{36qN}$, implying that (by Lemma 2R) no randomized queue-finding algorithm without a bad input can exist for sufficiently large $m$. This contradicts that this is a family of networks with randomized queue-finding algorithms free of bad inputs. We conclude that no such family exists.

It follows that, in either the determinstic or the randomized case, when the conditions of the theorem hold, we must have $qN \log d = \Omega(m \log m)$ in order for a family of bounded-degree networks with correct algorithms to exist. If local storage is constant, we must have $N \log d = \Omega(m \log m)$, so that the butterfly-based construction is optimal up to a constant factor. For $q = \Theta(\log m)$, the hypercube variation is not necessarily optimal due to is logarithmic degree.

# 4 Randomized Inputs

In order to study the effects of random inputs, we use a Poisson model of queue arrivals, where at each time step, the number of packets arriving (or being requested) at a port has a Poisson distribution with mean $\lambda$; note that this is a somewhat different and more general notion of arrival rate than was used before. Thus if $K$ is the number of packets arriving or departing at a port during one time step,

$$Pr(K = x) = \frac{e^{-\lambda}\lambda^x}{x!}$$

for $x = 0, 1, 2 \ldots$.

We will show that a group of $\log m$ ports will only with very small probability accumulate more than a total of $\log m$ packets In this case, we can solve each of the $\frac{m}{\log m}$ instances of this smaller problem with $q = \Theta(\log\log m)$, for a total of $\frac{m}{\log m}\Theta(\log m \log\log m) = \Theta(m \log\log m)$ space. This is much less than is required in the worst case, even if randomized algorithms are allowed.

# 5 Ongoing Research

The butterfly-based construction in section 2 shows that the lower bounds in section 3 are tight (to within a constant factor) when local storage is constant. However, $N = m \log m$ for this construction. Can a $q = \log m$, $N = m$ network with bounded degree and efficient space usage be found, or can the lower bounds be tightened to rule out such a possibilty?

Extension of our constructions to simulate random access local memories is not possible without a degradation in simulation time (i.e., real-time simulations are no longer possible), but it might be interesting if the results could be extended to simulate local memories that act like trees, if this is possible. Another area of research is to find the best achievable behavior for other specific networks — for instance, how many ports and packets can be managed in an $N$-node mesh?

# References


[BH] A. Borodin and J. Hopcroft, incomplete reference.

[K] B. Kuszmaul, personal communication, 1984.

[L] T. Leighton, Class notes in Theory of Parallel and VLSI Computation, 1984.

[PU] D. Peleg and E. Upfal, "The Token Distribution Problem," Proc. of 27th Ann. Symp on Foundations of Computer Science, 1986, pp 418-427.

[R] A. Ranade, "How to Emulate Shared Memory," Proc. of 28th Ann. Symp. on Foundations of Computer Science, 1987, pp 185-194.

[VB] L. G. Valiant and G. J. Brebner, "Universal Schemes for Parallel Computation," Proc. of 13th Ann. ACM Symp. on Theory of Computing, 1981, pp 263-277.

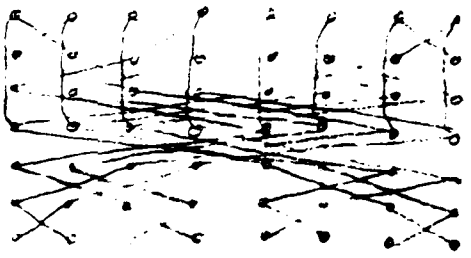



Figure 1

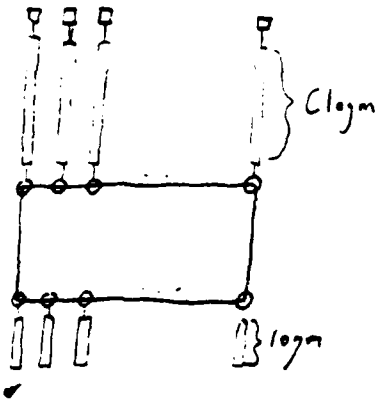



Figure 2



VLSI Memo No. 88-438
March 1988

# SOFTWARE STRUCTURING PRINCIPLES FOR VLSI CAD

Jacob Katzenelson and Richard Zippel

Abstract

VLSI CAD systems are typically large and undergo frequent changes. Such systems should be designed for reusability by anticipating change. Our thesis is that this goal can be achieved by designing the software as layers of problem oriented languages, which are implemented by suitably extending a "base" language. A language layer rarely needs to be adapted to changes, only the application (i.e. algorithm) needs to be changed. We present and compare two different implementations of this philosophy. The first uses UNIX and Enhanced C and the second uses Common Lisp on a Lisp machine. In each case, we describe the basic technique and its applications.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138